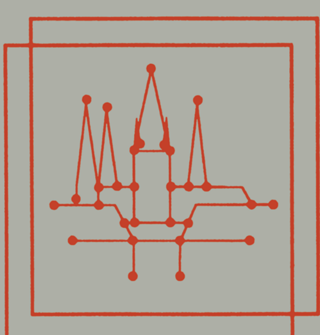
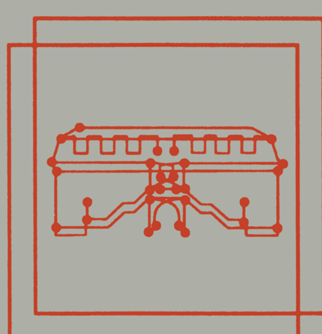
Lecture Notes in Computer Science 1731

Jan Kratochvíl (Ed.)

Graph Drawing

7th International Symposium, GD '99
Štiřín Castle, Czech Republic, September 1999
Proceedings





Springer

Lecture Notes in Computer Science 1731
Edited by G. Goos, J. Hartmanis and J. van Leeuwen

Springer
Berlin
Heidelberg
New York
Barcelona
Hong Kong
London
Milan
Paris
Singapore
Tokyo

Jan Kratochvíl (Ed.)

7th International Symposium, GD'99
Štiřín Castle, Czech Republic
September 15-19, 1999
Proceedings

Springer

Volume Editor

Jan Kratochvíl (Ed.)
Charles University, Faculty of Mathematics and Physics
School of Computer Science, Department of Applied Mathematics
118 00 Prague 1, Czech Republic
E-mail: honza@kam.ms.mff.cuni.cz

Cataloging-in-Publication data applied for

Die Deutsche Bibliothek - CIP-Einheitsaufnahme

Graph drawing : 7th international symposium ; proceedings / GD '99, Stírín Castle, Czech Republic, September 15 - 19, 1999 / Jan Kratochvíl (ed.). - Berlin ; Heidelberg ; New York ; Barcelona ; Hong Kong ; London ; Milan ; Paris ; Singapore ; Tokyo : Springer, 1999
(Lecture notes in computer science ; Vol. 1731)
ISBN 3-540-66904-3

CR Subject Classification (1998): G.2, I.3, F.2

ISSN 0302-9743
ISBN 3-540-66904-3 Springer-Verlag Berlin Heidelberg New York

Typesetting: Camera-ready by author
SPIN: 10704020 06/3142 – 5 4 3 2 1 0 Printed on acid-free paper

Preface

The range of issues considered in graph drawing includes algorithms, graph theory, geometry, topology, order theory, graphic languages, perception, applications, and practical systems. Much research is motivated by applications to systems for viewing and interacting with graphs. The interaction between theoretical advances and implemented solutions is an important part of the graph drawing field. The annually organized graph drawing symposium is a forum for researchers, practitioners, developers, and users working on all aspects of graph visualization and representations. The preceding symposia were held in Montreal (GD'98), Rome (GD'97), Berkeley (GD'96), Passau (GD'95), Princeton (GD'94), and Paris (GD'93).

The Seventh International Symposium on Graph Drawing GD'99 was organized at Štiřín Castle, in the vicinity of Prague, Czech Republic. This baroque castle recently restored as a hotel and conference center provided a secluded place for the participants, who made good use of the working atmosphere of the conference. In total the symposium had 83 registered participants from 16 countries.

The program committee had a hard time choosing the accepted 38 contributions from the 59 submitted ones. Every paper was read by at least 4 program committee members, carefully evaluated, and the final decision was made after a strenuous e-mail discussion. The accepted papers show the field of graph drawing in its broadness and versatility, with contributions ranging from theoretical results on graph representations and theory of graph drawing algorithms to system demonstrations of graph drawing software and practical applications. The poster gallery included only two presentations this year, both presented in the proceedings.

All three invited speakers provide strong links to the Prague school of Discrete Mathematics. Jiři Matoušek (Charles University, Prague) delivered a talk that showed a sample of techniques and tools used in designing geometric algorithms. Robin Thomas (Georgia Institute of Technology, Atlanta) presented a survey about planar graphs and related graph classes, including an overview of their recent computer aided re-proof of the Four Color Theorem. Both of these talks clearly showed the multiple connections of mathematics and practical computer science. Finally Jaroslav Nešetřil (Charles University, Prague), former advisor of both preceding speakers, presented a somewhat atypical talk. His multimedia show gave another dimension to the notion of 'drawing' and reflected the recent trends in the graph drawing area, where aestethic criteria continue to play an increasingly important role.

The proceedings are structured to reflect the program of the conference. In particular, all three invited talks are illustrated in the proceedings either by reprints of the transparencies or by an extended abstract.

The graph drawing contest was again the center of interest for the participants. This contest serves to monitor and to challenge the state of the art in the area of graph drawing. A contest report is included in the proceedings. Thanks to Joe Marks for organizing the contest and taking the time to put together the comprehensive report.

As a novelty a special prize was awarded to the best student contribution. Six of the accepted submissions were co-authored solely by students, and the program committee voted the best among these. The winner was David Wood for his paper 'Multi-Dimensional Orthogonal Graph Drawing with Small Boxes'.

Apart from the professional program, the participants warmly accepted a conference concert `stivin@stirin` given by one of the best known contemporary Czech jazz musicians Jiří Stivín and rewarded his one hour performance with a long lasting applause.

GD'99 was organized and sponsored by DIMATIA Charles University. This recently established Center for Discrete Mathematics, Theoretical Computer Science and Applications fosters and coordinates research in Discrete Mathematics in the Czech Republic, and also has strong international links. The center organizes series of workshops and hosts both short term visitors and long term postdoc programs. For more information see its Web page

`http://www.ms.mff.cuni.cz/acad/kam/dimatia/` .

Many people have contributed to the success of GD'99. First of all the authors, who submitted and presented high quality papers and system demonstrations. Program committee members invested a lot of their time into reading, evaluating, and selecting the accepted papers.

Thanks are due to all members of the organizing committee and to the volunteers (mostly students of Charles University). Special thanks to Jan Vondrák for organizing the poster gallery and to Vít Novák for making the computers and equipment ready when they were needed. But most special thanks are due to Jiří Fiala for managing the electronic submissions, collecting and proof-reading the accepted papers, and technically editing these proceedings, and to Anna Kotěšovcová from Conforg for excellent local organization. Without the help of these two special people the conference would have been just impossible.

Finally I want to thank the industrial sponsors for their financial or other help: Velké Popovice Brewery, Bohemia Glass, Sahm, Mitsubishi Electric Research Laboratory, AT&T Labs, and Tom Sawyer Software. The financial support from DIMATIA was partially made available by Czech Research grants GAČR 201/1999/0242 and Kontakt 1999/0338.

The coming Graph Drawing '00 will be organized by Joe Marks and Kathy Ryall in Williamsburg, Virginia, in September 2000.

September 1999 Jan Kratochvíl

Organization

GD'99 was organized by the

Program Committee

Giuseppe Di Battista (Rome)
Franz Brandenburg (Passau)
Hubert de Fraysseix (Paris)
Emden Gansner (AT&T)
Jan Kratochvíl (Chair, Prague)
Bill Lenhart (Williamstown)
Kim Marriott (Melbourne)
Joe Marks (MERL)
Bojan Mohar (Ljubljana)
Petra Mutzel (Saarbrücken)
Takao Nishizeki (Tohoku)
Kathy Ryall (Charlottesville)
Ioannis Tollis (Dallas)
Pavel Valtr (Prague)
Sue Whitesides (Montreal)

Steering Committee

Giuseppe Di Battista (Rome)
Franz Brandenburg (Passau)
Hubert de Fraysseix (Paris)
Peter Eades (Newcastle)
Jan Kratochvíl (Prague)
Joe Marks (MERL)
Takao Nishizeki (Tohoku)
Roberto Tamassia (Providence)
Ioannis Tollis (Dallas)
Sue Whitesides (Montreal)

Organizing Committee

Jiří Fiala
Vít Janota
Anna Kotěšovcová
Jan Kratochvíl
Jaroslav Nešetřil
Vít Novák
Pavel Valtr

Volunteers

Hana Čásenská
František Černohorský
Alena Fialová
Tomáš Chudlarský
Romana Jezdinská
Petra Kadlecová
Daniel Král
Jana Maxová
Jakub Šimek
Jan Vondrák

External Referees

E. Dahlhaus
W. Didimo
G. Farr
W. Feng
C. Gutwenger
M. Himsolt
M.-L. Huang
M. Kaufmann
G. Klau
X. Lin
G. Liotta
K. Miura
S. Nakano
J. Nešetřil
P. Ossona de Mendez
M. Patrignani
M. Pizzonia
S. Rahman
F. Schreiber
R. Webber
R. Weiskircher
D. Wood
T. Ziegler

Table of Contents

Invited Talk

Orthogonality I

Levels I

Clusters I

Drawing I

Invited Talk

Planarity

Clusters II

Levels II

Applications

Invited Talk

Symmetry

Orthogonality II

Posters

The Anatomy of a Geometric Algorithm*

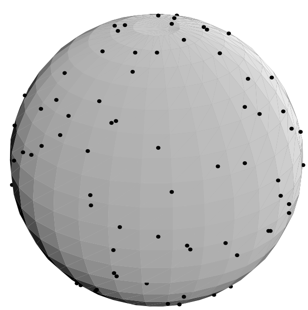
Jiří Matoušek

Department of Applied Mathematics, Charles University
Prague, Czech Republic
matousek@kam.ms.mff.cuni.cz

What is computational geometry?

(examples rather than a definition)

An n-point set in $\mathbf{R}^3$:



- What is the diameter of this point set?
- What is the smallest ball enclosing it?
- What is the "best fit" by a sphere?
- What is a "nice surface" defined by these points?

$\vdots$

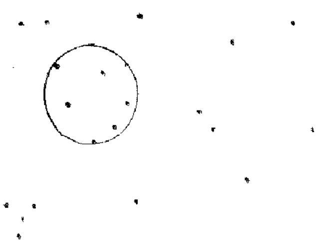
* This article contains the speaker's slides for the talk, reproduced by the editors with speaker's permission. An appendix with references was prepared by the speaker.

J. Kratochvíl (Ed.): GD'99, LNCS 1731, pp. 1–7, 1999.

OUR PARTICULAR PROBLEM:

INPUT: n POINTS in the plane, $k \le n$

F(M): A SMALLEST CIRCLE CONTAINING k POINTS



WHILE DISCUSSING AN ALGORITHM, WE WILL MEET

... TECHNIQUES

- Reformulation in terms of arrangements
- simple combinatorial-geometric observations
- randomized algorithms
- parametric search (and how to avoid it)
- "n^2-hard" problems
- geometric optimization framework (LP-type problems)

• TYPICAL OBSTACLES TO IMPLEMENTATION

TRICK: LOOK AT FIXED SIZE PROBLEM

Given n points and $r \ge 0$, is there a circle of radius r enclosing $\ge k$ points?

"TESTING" given r

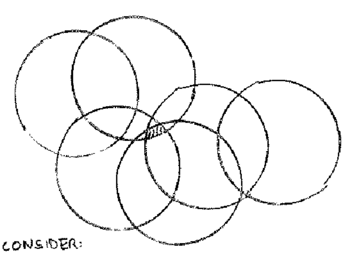
"CONVERSION" BY GENERAL TECHNIQUES (PARAMETRIC SEARCH & OTHERS) TO

"SEARCHING" for $\min\{r \ge 0 : \text{test positive}\}$

REFORMULATION WITH ARRANGEMENTS

$\exists$ r-DISC CONTAINING $\ge k$

$\Longleftrightarrow$

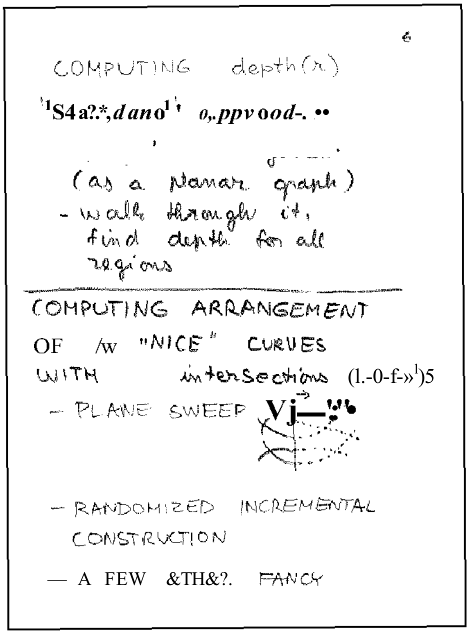


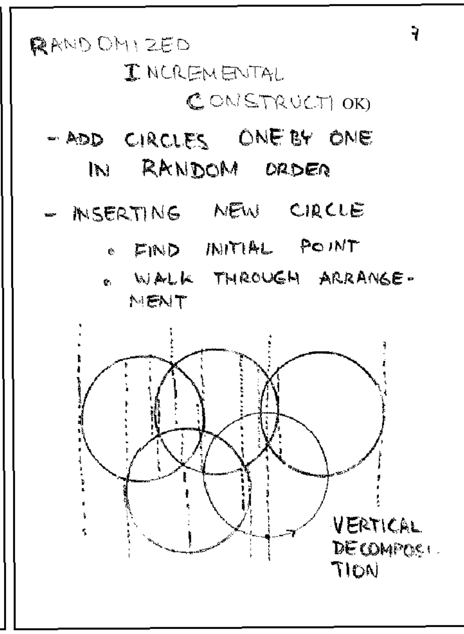
CONSIDER:

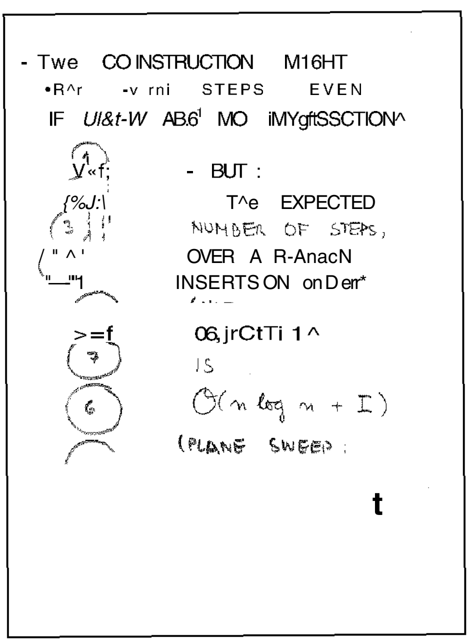
n disks of radius r around the given points

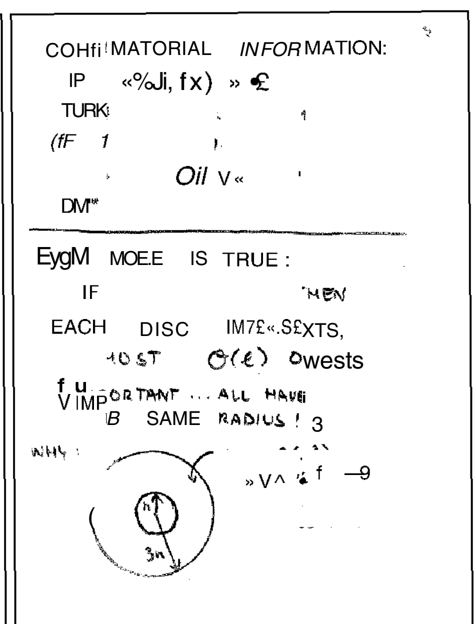
$\exists$ point of DEPTH $\ge k$

(depth = # of containing discs)


COMPUTING depth(x)
(as a planar graph)
- walk through it, find depth for all regions
COMPUTING ARRANGEMENT
OF "NICE" CURVES
WITH intersections
- PLANE SWEEP
- RANDOMIZED INCREMENTAL CONSTRUCTION
FANCY


RANDOMIZED INCREMENTAL CONSTRUCTION
- ADD CIRCLES ONE BY ONE IN RANDOM ORDER
- INSERTING NEW CIRCLE
• FIND INITIAL POINT
• WALK THROUGH ARRANGEMENT
VERTICAL DECOMPOSITION


STEPS EVEN
- BUT :
EXPECTED
NUMBER OF STEPS,
OVER A
IS
O(n log n + I)
(PLANE SWEEP


INFORMATION:
IS TRUE :
IF
THEN
EACH DISC
O(k)
IMPORTANT ... ALL HAVE
SAME RADIUS !
WHY :

MCVM Vb£' C ftM TKT

GIVEN r (COMPUTE DEPTH(r))

MOW Do \wE' F/M]>>

"S"?ftNt>ARX)" C Cl,$r\)&& !!(") APPROACH:

PARAMETRIC SEARCH

[_D g ^ j ^ TAltE SOMF AlGtmiTNH G

SUCH AS: test depth$(r) \geq k$

SIMULATE t n ^ Exrcuno/v

r>p ^ PoK. ^ = H *

$r^* \lesseqgtr q$ ← given

depth$(q) < k \Rightarrow r^* > q$

depth$(q) \geq k \Rightarrow r^* \leq q$

[ignore equality for now]

TO RESOLVE 1 COMPARISON $r \overset{?}{\sim} q$

NEED TO COMPUTE depth(q) 1x

^ ^} i & ^ A'NV COM ^rRtfe 0 MJ5.

)Vr ONCE C*

SEA rt CN

⟶

PARAM. SEARCH ALGORITHMS

ASVNPT. QUITE FAbT (i.oes ioiT) ONLY

PfsRftM • SfftfStM ETC'

n) algorithm

(probably impractical)

OTHER AppaoACMFs.:

— kth ORDER VOliOMo I DIAGRAMS

SUBDIVISION OP TUB' PLANE

REGION = POINTS WITH THE

SAME k-tuple os-

^tAa.'R,T POTOTi IN 7WP

GIVEN SET

KNOWN complexity

CONSTRUCTION TIME ctose

TO $O(k(n-k))$ [log n]

— ANALYZE "EVENTS" IN PARAM. SEARCH,

(PflMvH. EACCH BtCLfttEP B'i

RANDOMIZED SEARCH - PROBLEM-SPECIFIC)

PHASE 1

FINO r_0 S i * WITH

g; ∸ 1ST OF Oft) POTBK'TIA'L

TRICK: APPROXIMATE depth(r)

BY grid-depth(r)

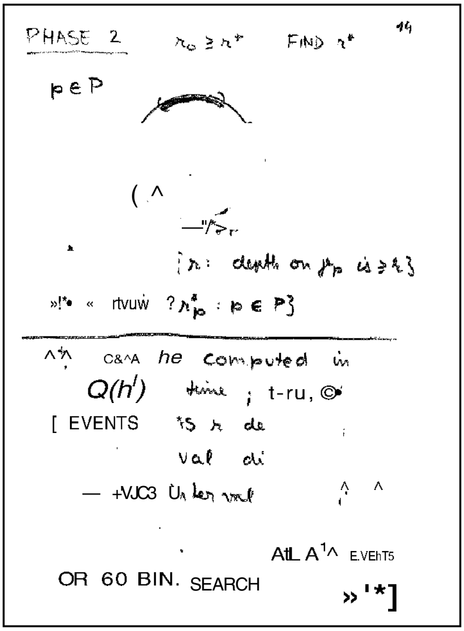
volume argument:

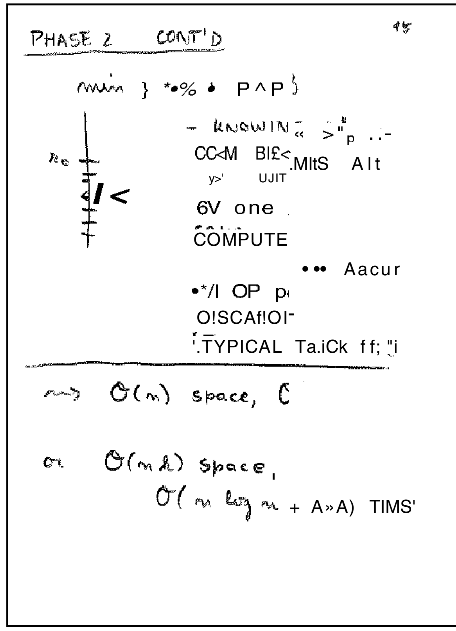
grid-depth$(r) \leq$ depth(r)

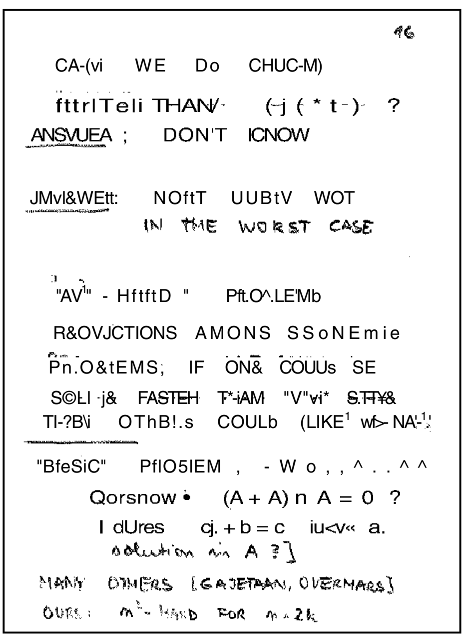
$\leq$ const · grid-depth(r)

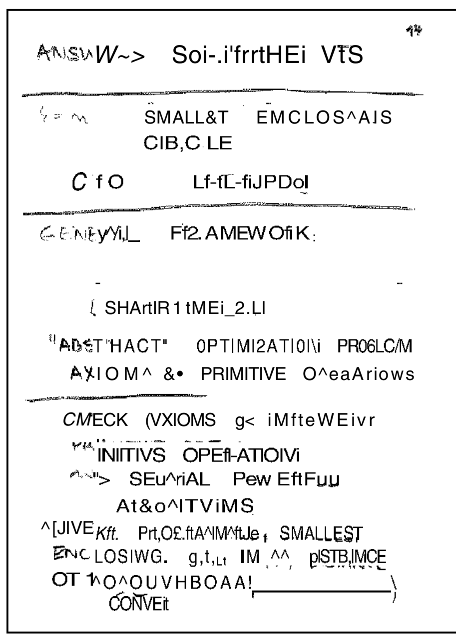
B.ESUUT-, PHASE 1 CAM &e DONE

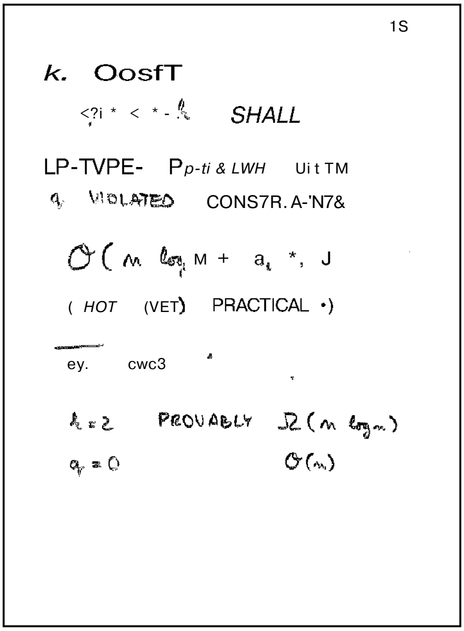
IN $O(n$ ℓKih) TIME S- 6f→) SPACE


14
PHASE 2
FIND
p ∈ P
{ r : depth on p is ≥ k }
rtvuw
AtL A EVEhT5
OR 60 BIN. SEARCH
he computed in
time ; t-ru,
[EVENTS
val
+VJC3

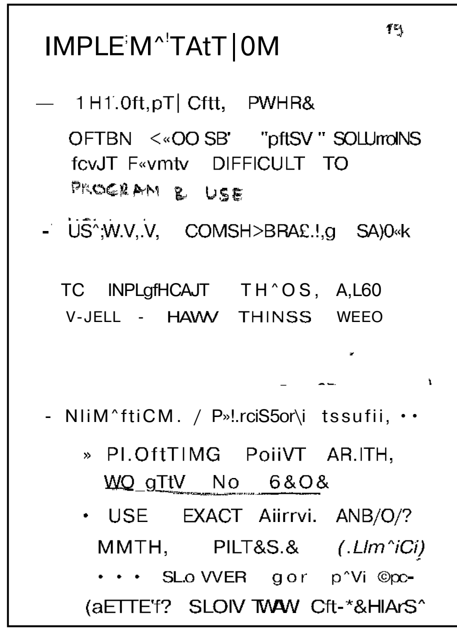

15
PHASE 2 CONT'D
P^P
KNOWIN
CC<M BI£<.MltS Alt
6V one
COMPUTE
Aacur
OP
O!SCAf!OI
TYPICAL Ta.iCk
O(n) space,
or O(n k) space,
O(n log n + A»A) TIMS'


16
CA-(vi WE Do CHUC-M)
fttrlTeli THAN/ (-j (* t -) ?
ANSVUEA ; DON'T ICNOW
JMvl&WEtt: NOftT UUBtV WOT
IN THE WORST CASE
"AV" - HftftD " Pft.O^.LE'Mb
R&OVJCTIONS AMONS SSoNEmie
Pn.O&tEMS; IF ON& COUUs SE
S©LI j& FASTEH T*iAM "V"vi* S.TH¥&
TI-?B\i OThB!.s COULb (LIKE wb NA-
"BfeSiC" PflO5IEM , - W o , , ^ . . ^ ^
Qorsnow (A + A) n A = 0 ?
I dUres cj. + b = c iu<v« a.
solution in A ?]
MANY OTHERS [GAJETAAN, OVERMARS]
OURS: n - HARD FOR n - 2k

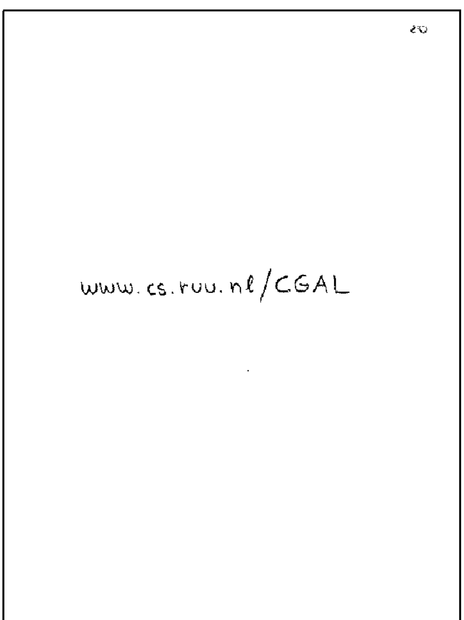

17
ANSWER Soi-.i'frrtHEi VTS
SMALL&T EMCLOS^AIS
CIB,C LE
C f O Lf-fL-fiJPDol
GENERAL Ff2. AMEWOfiK:
SHArtIR 1 tMEi_2.LI
"ABST'HACT" 0PTIMI2ATI0I\i PR06LC/M
AXIOM^ &• PRIMITIVE O^eaAriows
CHECK (VXIOMS g< iMfteWEivr
INIITIVS OPEfl-ATIOIVi
SEu^riAL Pew EftFuu
At&o^ITViMS
^[JIVE Kft. Prt,O£.ftA^IM^ftJe, SMALLEST
ENCLOSIWG. g,t,Lt IM pISTB,IMCE
OT ^O^OUVHBOAA!
CONVEit



1S
k. OosfT
SHALL
LP-TVPE- Pp-ti & LWH Uit TM
VIOLATED CONS7R.A-'N7&
(HOT (VET) PRACTICAL •)
ey. cwc3
k = 2 PROVABLY
q = 0



IMPLEM^TAtT|0M
— 1H1.0ft,pT| Cftt, PWHR&
OFTBN <«OO SB' "pftSV " SOLUrrolNS
fcvJT F«vmtv DIFFICULT TO
PROGRAM & USE
- US^;W.V,.V, COMSH>BRA£.!,g SA)0«k
TC INPLgfHCAJT TH^OS, A,L60
V-JELL - HAVW THINSS WEEO
- NIiM^ftiCM. / P»!.rciS5or\i tssufii, ••
» PI.OftTIMG PoiiVT AR.ITH,
WO gTtV No 6&O&
• USE EXACT Aiirrvi. ANB/O/?
MMTH, PILT&S.& (.LIm^iCi)
• • • SLo VVER gor p^Vi ©pc-
(aETTE'f? SLOIV TWAW Cft-*&HIArS^



www.cs.ruu.nl/CGAL

Appendix

The purpose of this small appendix to the slides is to cite sources for the results directly mentioned in the talk, and to point to a few general-purpose references on computational geometry.

There are several introductory textbooks by now; one of them is [2]. Randomized incremental algorithms and applications of arrangements can be found there. There are two handbooks of computational geometry [6], [11]. Recent activity in the field, with increasing emphasis on more practically oriented studies, can be monitored in proceedings of the Annual ACM Symposia on Computational Geometry.

A parametric search algorithm for the considered problem is from [3]. Parametric search was formulated in [10]. Other papers on the problem are [4], [1]. The algorithm discussed in some detail is from [9]. The N^2-hard problems are collected in [5]. The LP-type problems were introduced in [8], and the application on circles enclosing all but q points is in [7].

References

1. A. Datta, H.-P. Lenhof, C. Schwarz, and M. Smid. Static and dynamic algorithms for k-point clustering problems. In *Proc. 3rd Workshop Algorithms Data Struct.*, volume 709 of *Lecture Notes Comput. Sci.*, pages 265–276. Springer-Verlag, 1993.
2. M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications.* Springer-Verlag, Berlin, 1997.
3. A. Efrat, M. Sharir, and A. Ziv. Computing the smallest k-enclosing circle and related problems. *Comput. Geom. Theor. Appl.*, 4:119–136, 1994.
4. D. Eppstein and J. Erickson. Iterated nearest neighbors and finding minimal polytopes. In *Proc. 4th ACM-SIAM Sympos. Discrete Algorithms*, pages 64–73, 1993.
5. A. Gajentaan and M. H. Overmars. On a class of n^2-hard problems in computational geometry. *Comput. Geom. Theor. Appl.*, 5(3):117–142, 1995.
6. J. E. Goodman and J. O'Rourke, editors. *Handbook of Discrete and Computational Geometry.* CRC Press LLC, Boca Raton, FL, 1997.
7. J. Matoušek. On geometric optimization with few violated constraints. *Discrete Comput. Geom.*, 14:365–384, 1995.
8. J. Matoušek, M. Sharir, and E. Welzl. A subexponential bound for linear programming. *Algoritmica*, 16:498–516, 1996.
9. J. Matoušek. On enclosing k points by a circle. *Information Processing Letters*, 53:217–221, 1995.
10. N. Megiddo. Applying parallel computation algorithms in the design of serial algorithms. *Journal of the ACM*, 30:852–865, 1983.
11. J.-R. Sack and J. Urrutia, editors. *Handbook on Computational Geometry.* North-Holland, 1999. In press.

Turn-Regularity and Planar Orthogonal Drawings*

(Extended Abstract)

Stina S. Bridgeman[1], Giuseppe Di Battista[2], Walter Didimo[2], Giuseppe Liotta[3], Roberto Tamassia[1], and Luca Vismara[1]

[1] Center for Geometric Computing, Department of Computer Science, Brown University, 115 Waterman Street, Providence, RI 02912–1910.
{ssb,rt,lv}@cs.brown.edu
[2] Dipartimento di Informatica e Automazione, Università degli Studi di Roma Tre, Via della Vasca Navale 79, 00146 Roma, Italy.
{gdb,didimo}@dia.uniroma3.it
[3] Dipartimento di Ingegneria Elettronica e dell'Informazione, Università degli Studi di Perugia, Via Duranti 93, 06131 Perugia, Italy.
liotta@diei.unipg.it

Abstract. Given an orthogonal representation H with n vertices and bends, we study the problem of computing a planar orthogonal drawing of H with small area. This problem has direct applications to the development of practical graph drawing techniques for information visualization and VLSI layout. In this paper, we introduce the concept of turn-regularity of an orthogonal representation H, provide combinatorial characterizations of it, and show that if H is turn-regular (i.e., all its faces are turn-regular), then a planar orthogonal drawing of H with minimum area can be computed in $O(n)$ time, and a planar orthogonal drawing of H with minimum area and minimum total edge length within that area can be computed in $O(n^{7/4} \log n)$ time. We also apply our theoretical results to the design and implementation of new practical heuristic methods for constructing planar orthogonal drawings. An experimental study conducted on a test suite of orthogonal representations of randomly generated biconnected 4-planar graphs shows that the percentage of turn-regular faces is quite high and that our heuristic drawing methods perform better than previous ones.

1 Introduction

Orthogonal drawings are drawings of graphs in which every edge is represented by a chain of horizontal and vertical segments. An orthogonal representation is an equivalence class of orthogonal drawings that have the same "shape" (see

* Research supported in part by the Consiglio Nazionale delle Ricerche under Project "Geometria Computazionale Robusta con Applicazioni alla Grafica ed al CAD", by the National Science Foundation under grants CCR-9732327 and CDA-9703080, and by the U.S. Army Research Office under grant DAAH04-96-1-0013.

J. Kratochvíl (Ed.): GD'99, LNCS 1731, pp. 8–26, 1999.

Fig. 1). This class is formally described by specifying the bends along each edge and the angles between consecutive edges around each vertex. In this paper we consider planar orthogonal representations, that is, equivalence classes of orthogonal drawings for which at least one of the drawings is planar. Given a planar orthogonal representation H, the problem of finding a planar orthogonal grid drawing of H with small area is usually referred to as the *compaction* of H.

Orthogonal representations and planar orthogonal drawings have been extensively investigated (see, e.g., [1, 9, 11, 12, 14, 15, 16, 18, 27, 28, 29, 30]) because of their direct applications to the development of practical graph drawing techniques for information visualization [6]. In particular, it has been experimentally shown that drawing algorithms for general graphs based on the compaction of orthogonal representations with minimum number of bends perform better in practice than other known orthogonal drawing algorithms [7, 25]. Orthogonal representations and related concepts, such as slicing floorplans, are also widely used in VLSI layout compaction algorithms (see, e.g., [19, 21, 22, 26, 31]).

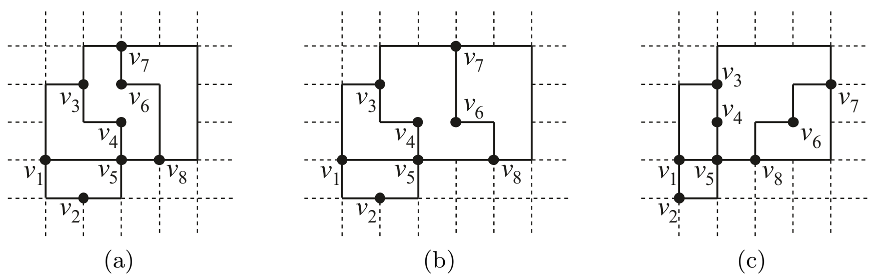


Fig. 1. Three planar orthogonal drawings of a graph. Two drawings of the same orthogonal representation are shown in (a) and (b), while a drawing of a different orthogonal representation is shown in (c). The drawing in (a) has optimal area among all planar drawings of that orthogonal representation.

Despite the significant body of research on orthogonal representations, the development of effective compaction techniques remains a challenging task. It has been conjectured for a long time [30], and recently proved [24], that the optimal compaction of planar orthogonal representations, i.e., computing a minimum area planar orthogonal grid drawing of a given planar orthogonal representation, is an NP-complete problem. The only class of planar orthogonal representations for which a polynomial-time optimal compaction algorithm is known is the trivial class of orthogonal representations whose faces are all rectangular [6].

From a practical perspective, the compaction algorithms used by current graph drawing libraries and systems, such as *AGD*[1], *GDToolkit*[2], and the *Graph Drawing Server*[3], are all variations of the compaction technique presented in [18,

[1] `http://www.mpi-sb.mpg.de/AGD/`
[2] `http://www.dia.uniroma3.it/~gdt/`
[3] `http://www.cs.brown.edu/cgc/graphserver/`

27], which is based on the idea of splitting faces into rectangles. Since the splitting imposes unnecessary constraints on the geometry, the resulting drawings may have substantially suboptimal area.

The importance of compaction techniques for graph visualization applications is confirmed by a recent work of Klau and Mutzel [20]. They consider the problem of assigning coordinates to vertices and edge bends of an orthogonal representation so that the total edge length is minimized. The problem is formulated as an integer linear program, whose practical performance is fairly good. Also, they show that the problem can be solved in polynomial time for those orthogonal representations in which there is only one possible relative position of any two vertices that results in a planar drawing; in this case, the inequalities of the corresponding ILP formulation form a totally unimodular matrix. The problem of minimizing the area of the drawing is not considered.

The main results of this paper can be summarized as follows.

- Given a planar orthogonal representation H, we define the concept of *turn-regularity* of a face of H, which is based on the structure of the sequence of left and right turns encountered when traversing the face. We show that the turn-regularity of a face can be tested in linear time.
- We relate turn-regularity to the concept of *switch-regularity* [8]. Namely, we characterize the turn-regularity of a face f in terms of the switch-regularity of two upward orientations of f.
- We introduce the concept of *orthogonal relation* between two vertices of H. This relation establishes the relative position of the two vertices in any planar orthogonal drawing of H. We show that an orthogonal relation is defined between every two vertices of H if and only if all the faces of H are turn-regular.
- We show that if H is turn-regular (i.e., all its faces are turn-regular), then any orthogonal drawing of H such that the orthogonal relations between every two vertices are satisfied is planar.
- We show that if H is turn-regular, then a planar orthogonal drawing of H with optimal area can be computed in $O(n)$ time and space, where n is the number of vertices and bends of H. Furthermore, a planar orthogonal drawing of H with optimal area and minimum total edge length within that area can be computed in $O(n^{7/4} \log n)$ time.
- We present the results of an experimental study on a test suite of planar orthogonal representations of randomly generated biconnected 4-planar graphs. The experiments show that the percentage of turn-regular faces is quite high (the average value is 89%). Motivated by this result, we have designed compaction heuristics based on the idea of "face turn-regularization". Namely, we decompose non-turn-regular faces into turn-regular ones, and then perform an optimal compaction of the resulting planar orthogonal representation. We implemented our compaction algorithms and experimentally observed that the improvement in area is substantial when compared to the compaction algorithms available in state-of-the-art graph drawing libraries.

Due to space limitations, some technical lemmas, the proofs, and some figures are omitted in this extended abstract. A full version of this paper is available on-line at `ftp://ftp.cs.brown.edu/pub/techreports/99/cs99-04.ps.Z`.

2 Preliminaries

2.1 Basic Definitions

We assume familiarity with graph terminology and basic properties of planar graphs (see, e.g., [5, 17]). The graphs we consider are assumed to be connected. For background on graph drawing, see [6].

An *st-digraph* is an acyclic digraph with a single source s (vertex with no incoming edges) and a single sink t (vertex with no outgoing edges). A *planar st-digraph* is an st-digraph that is planar and embedded with vertices s and t on the boundary of the external face. An important property of planar st-digraphs is that the incoming edges of each vertex v appear consecutively around v, as do the outgoing edges. Also, the boundary of each face f consists of two directed paths enclosing f, with common origin and destination.

A planar graph whose vertices have degree at most four is said to be *4-planar*. Let G be an embedded 4-planar graph and let f be a face of G. In the following, we always traverse the boundary of f so that f is on the left, i.e., counterclockwise if f is internal and clockwise if f is external. The boundary of f consists of an alternating circular sequence of vertices and edges. Note that if G is not biconnected, there may be two occurrences of the same edge and multiple occurrences of the same vertex on the boundary of f. We denote by a_f the number of vertices (or edges) of f, each counted with its multiplicity.

Informally speaking, an *orthogonal representation* of an embedded 4-planar graph G describes an equivalence class of orthogonal drawings of G with "similar shape". It consists of a "decorated" version of G where each pair of consecutive edges around a vertex v is assigned an angle multiple of $\pi/2$ and each edge (u, v) is assigned a sequence of bends going from u to v, each a left or right turn. In this paper we consider *planar orthogonal representations*, that is, equivalence classes of orthogonal drawings for which at least one of the drawings is planar; in the rest of the paper, we omit the word planar when referring to orthogonal representations. For a detailed definition of orthogonal representation, see, e.g., [6].

Since each bend can be replaced by a dummy vertex of degree 2, in the rest of the paper we assume, for the sake of simplicity, that orthogonal representations have no bends. We also assume that different drawings of the same orthogonal representation are iso-oriented, i.e., each edge has the same direction and its end-vertices are in the same relative position.

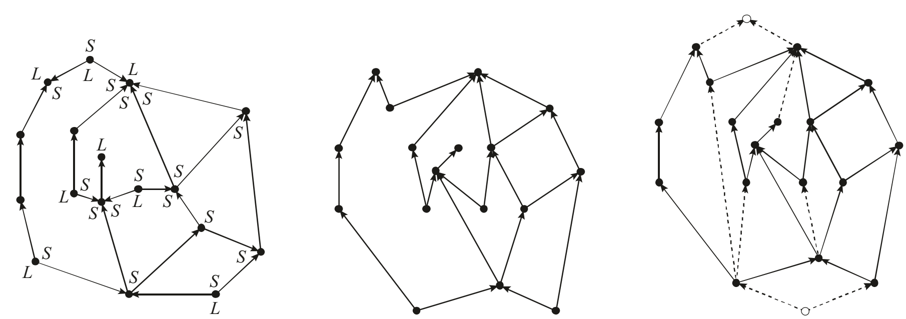
2.2 Switch-Regularity

We recall some terminology and results from [2, 8, 10]. A drawing of a digraph is said to be *upward* if edges are mapped to curves monotonically increasing in a

common direction, for instance the vertical one. A digraph is *upward planar* if it admits an upward planar drawing. As we are going to show in the next section, upward planar drawings and orthogonal representations are strictly related. We recall here some notations and results that will be useful in the rest of the paper.

A vertex v of an embedded planar digraph G is said to be *bimodal* if all the incoming edges of v appear consecutively around it in the embedding, and so do the outgoing edges. If all the vertices of G are bimodal then G and its embedding are called *bimodal*. Let f be a face of a bimodal embedded planar digraph G. A vertex v of f with incident edges e_1 and e_2 on the boundary of f is a *switch* of f if e_1 and e_2 are both incoming or both outgoing edges (note that e_1 and e_2 may coincide if the digraph is not biconnected). In the former case v is a *sink-switch* of f, in the latter a *source-switch* of f. Observe that a source (sink) of G is a source-switch (sink-switch) of all its incident faces; a vertex of G that is not a source or a sink is a switch of all its incident faces except two. We denote by $2n_f$ the number of switches of f.

Assign S and L labels to the switches of each face of a bimodal embedded planar digraph G such that (see Fig. 2a): (i) each source or sink of G has exactly one L label; (ii) for each face f, the number of L-labeled switches assigned to f is equal to $n_f - 1$ if f is an internal face, and to $n_f + 1$ if f is the external face. The S-labeled (L-labeled) source-switches are called s_S-switches (s_L-switches) and the S-labeled (L-labeled) sink-switches are called t_S-switches (t_L-switches). The circular sequence of labels of f so obtained is a *labeling* of f and is denoted by σ_f. Also, S_{σ_f} (L_{σ_f}) denotes the number of S-labels (L-labels) of σ_f. A face f of G labeled in this manner is *upward consistent*.



(a) (b) (c)

Fig. 2. (a) A bimodal embedded planar digraph G with an upward consistent labeling of its faces. (b) An upward planar drawing of G corresponding to the upward consistent labeling in (a). (c) A complete saturator of G; s and t are represented as white circles, and saturating edges are represented as dashed segments.

Theorem 1. [2] *A bimodal embedded digraph is upward planar if and only if all its faces have an upward consistent labeling.*

Let G be a bimodal embedded digraph such that all its faces have an upward consistent labeling. For each face of G, the S-label (L-label) assigned to a switch intuitively indicates that the angle formed by the two edges identifying the switch is smaller (larger) than π in an upward planar drawing of G. Any such drawing is said to *correspond* to the upward consistent labeling of the faces of G. On the other hand, given an upward planar drawing of a bimodal embedded digraph G, an upward consistent labeling for each face of G can be obtained by simply checking whether the angle formed by each pair of edges identifying a switch is smaller or larger than π. Figure 2b shows an upward planar drawing (corresponding to the upward consistent labeling) of the embedded planar digraph in Fig. 2a.

Given an embedded upward planar digraph G, a *saturator* of G is a set of vertices and edges, not belonging to G, with which we augment G. More precisely, a saturator consists of two vertices s and t, edge (s, t), and a set of edges (u, v) (each edge a *saturating edge*) such that:

– Vertices u and v are switches of the same face, or $u = s$ and v is an s_L-switch of the external face, or u is a t_L-switch of the external face and $v = t$.
– If, $u, v \neq s, t$, either u is an s_S-switch and v is an s_L-switch or u is a t_L-switch and v is a t_S-switch. In the former case we say that u *saturates* v and in the latter case we say that v *saturates* u.
– The faces obtained with the insertion of a saturating edge are upward consistent.

A saturator of G is said to be *complete* if for every face f and for every switch u of f labeled L, u is an end-vertex of an edge of the saturator (see Fig. 2c). Clearly, adding to G a complete saturator yields a planar st-digraph.

Lemma 1. [8] *Every upward planar embedding admits a complete saturator.*

An upward planar embedding may have, in general, several complete saturators. The class of embedded upward planar digraphs for which there exists a unique complete saturator has been characterized in [8]. The characterization is based on a certain type of labeling. Namely, let G be an embedded upward planar digraph. An internal face f of G has a *switch-regular labeling* if σ_f does not contain two distinct maximal subsequences σ_1 and σ_2 of S-labels such that $S_{\sigma_1} > 1$ and $S_{\sigma_2} > 1$. An external face f of G has a switch-regular labeling if σ_f does not contain two consecutive S-labels. (Note that a switch-regular labeling is called just regular labeling in [8].) A face of G with a switch-regular labeling is a *switch-regular* face. For example all faces of Fig. 2a are switch-regular. An embedded upward planar digraph is *switch-regular* if all its faces have a switch-regular labeling. The corresponding embedding is also called *switch-regular*.

Theorem 2. [8] *An upward planar embedding has a unique complete saturator if and only if it is switch-regular.*

3 Turn-Regularity and Switch-Regularity

3.1 Orthogonal Relations

Let G be an embedded 4-planar graph, H be an orthogonal representation of G, Γ be a planar drawing of H, and v be a vertex of G. We denote by $x(v)$ and $y(v)$ the x- and y-coordinates of the point representing v in Γ. We define four binary relations on the vertex set of G: for each pair $\{u, v\}$ of vertices of G, these relations determine the relative position of u and v in *all planar drawings* of H.

- $u <_x v$ if $x(u) < x(v)$ for all planar drawings of H; in this case, we say that u is *left* of v and that v is *right* of u.
- $u =_x v$ if $x(u) = x(v)$ for all planar drawings of H; in this case, we say that u is *x-aligned* with v.
- $u <_y v$ if $y(u) < y(v)$ for all planar drawings of H; in this case, we say that u is *below* v and that v is *above* u.
- $u =_y v$ if $y(u) = y(v)$ for all planar drawings of H; in this case, we say that u is *y-aligned* with v.

We refer to the first two binary relations as x-relations and to the second two binary relations as y-relations. As an example, in the orthogonal representation in Fig. 1 $v_2 <_x v_8$, $v_6 =_x v_7$, $v_2 <_y v_3$, and $v_1 =_y v_5$.

We define three new binary relations on the vertex set of G, obtained by combining an x-relation and a y-relation: $=_x \wedge <_y$, $<_x \wedge =_y$, and $<_x \wedge <_y$. These three binary relations together with the binary relations $<_x$ and $<_y$ are collectively referred to as *orthogonal relations*. As an example, in the orthogonal representation in Fig. 1 $v_5 =_x v_4 \wedge v_5 <_y v_4$, $v_1 <_x v_8 \wedge v_1 =_y v_8$, and $v_1 <_x v_7 \wedge v_1 <_y v_7$, while no orthogonal relation holds for $\{v_4, v_6\}$.

3.2 Turn-Regularity

To characterize those orthogonal representations that have an orthogonal relation for each pair of vertices, we introduce the notion of *turn-regularity*.

Let G be an embedded 4-planar graph, H be an orthogonal representation of G, and f be a face of G. For each occurrence of vertex v on the boundary of f, let *prev*(v) and *next*(v) be the edges preceding and following v, respectively, on the boundary of f. Note that *prev*(v) and *next*(v) may coincide if the graph is not biconnected. We associate with each occurrence of v one or two *corners*. Namely:

- If the angle internal to f between *prev*(v) and *next*(v) is $\pi/2$ in H, we associate with v one *convex* corner, and say that v corresponds to a *left turn*.
- If the angle internal to f between *prev*(v) and *next*(v) is π in H, we associate with v one *flat* corner, and say that v corresponds to a *flat turn*.
- If the angle internal to f between *prev*(v) and *next*(v) is $3\pi/2$ in H, we associate with v one *reflex* corner, and say that v corresponds to a *right turn*.

- If the angle internal to f between $prev(v)$ and $next(v)$ is 2π in H, we associate with v an ordered pair of *reflex* corners, and say that v corresponds to a *U-turn*.

Hence, a circular sequence of corners can be associated with the boundary of f. For each corner c of f, let $turn(c) = 1$ if c is convex, $turn(c) = 0$ if c is flat, and $turn(c) = -1$ if c is reflex. As an example, in Fig. 3 a convex corner is associated with v_1, a flat corner with v_2, a reflex corner with v_3, and an ordered pair of reflex corners with v_4. The grey portion of Fig. 3 represents the rest of the graph.

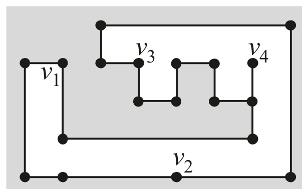


Fig. 3. A face of an orthogonal representation. The grey portion represents the rest of the graph.

For each ordered pair $\{c_i, c_j\}$ of corners associated with vertices of f, let $rotation(c_i, c_j) = \sum_c turn(c)$ for all corners c along the boundary of f from c_i (included) to c_j (excluded). If c_i and c_j are associated with distinct vertices v_i and v_j, respectively, $rotation(c_i, c_j) \cdot \pi/2$ is the net angle turned between $prev(v_i)$ and $prev(v_j)$. As an example, in Fig. 3 let c_1, c_2, and c_3 be the corners associated with v_1, v_2, and v_3, respectively, and let $\{c'_4, c''_4\}$ be the ordered pair of corners associated with v_4; $rotation(c_1, c_2) = 3$, $rotation(c_3, c'_4) = 1$, $rotation(c_3, c''_4) = 0$, and $rotation(c_3, c_1) = -3$.

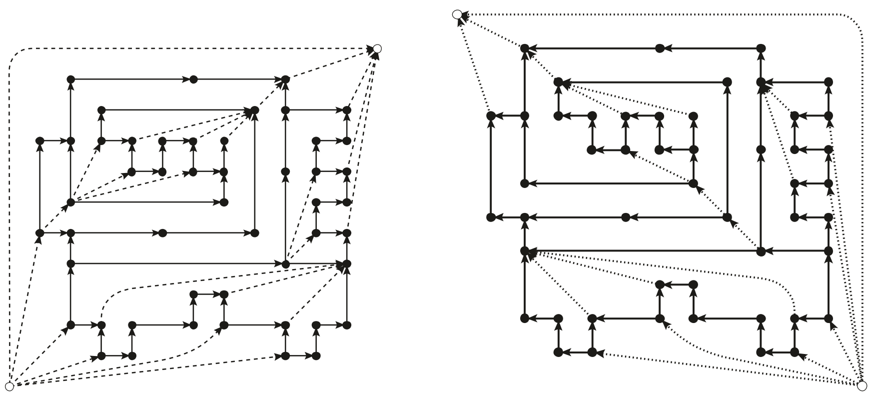
Two reflex corners c_i and c_j are called *kitty corners* if $rotation(c_i, c_j) = 2$ or $rotation(c_j, c_i) = 2$. In Fig. 1, the corners associated with vertices v_4 and v_6 are kitty corners. A face of an orthogonal representation is *turn-regular* if it has no kitty corners. As an example, the face shown in Fig. 3 is turn-regular. An orthogonal representation is *turn-regular* if all its faces are turn-regular.

3.3 Turn-Regularity and Switch-Regularity

Let G be an embedded 4-planar graph, H be an orthogonal representation of G, and Γ be a planar drawing of H. Let Γ_r be an orientation of Γ such that all vertical segments are directed upward and all horizontal segments are directed rightward, and let Γ_ℓ be an orientation of Γ such that all vertical segments are directed upward and all horizontal segments are directed leftward. Observe that Γ_r is an upward planar drawing in the North-East direction and that Γ_ℓ is an upward planar drawing in the North-West direction. Γ_r and Γ_ℓ induce two orientations on H. We denote the oriented orthogonal representations by

H_r and H_ℓ. In turn, H_r and H_ℓ induce two orientations on G. We denote the embedded 4-planar digraphs by G_r^H and G_ℓ^H, respectively. Observe that G_r^H and G_ℓ^H are embedded upward planar digraphs. Also, note that different orthogonal representations of G induce, in general, different orientations on G; since we work with a fixed orthogonal representation of a graph, we use G_r and G_ℓ in the rest of the paper, omitting the reference to H.

Theorem 3. *An orthogonal representation H of an embedded 4-planar graph G is turn-regular if and only if the embedded upward planar digraphs G_r and G_ℓ are both switch-regular.*



(a) (b)

Fig. 4. (a) G_r (edges represented as solid segments) and a complete saturator (edges represented as dashed segments). (b) G_ℓ (edges represented as solid segments) and a complete saturator (edges represented as dotted segments).

4 Orientations and Paths

Let G be an embedded 4-planar graph and H be a turn-regular orthogonal representation of G. As seen in Section 2.2, a complete saturator of an embedded upward planar digraph consists of two vertices s and t and a set of (directed) saturating edges. Figure 4a shows a complete saturator of graph G_r corresponding to the oriented orthogonal representation H_r. Figure 4b shows a complete saturator of graph G_ℓ corresponding to the oriented orthogonal representation H_ℓ. In the rest of the paper we never consider the saturating edges of G_r and G_ℓ incident with s or t, even when not explicitly stated. Let f be an internal face of H; a maximal vertical or horizontal chain of f is said to be *unconstrained* if both its end-vertices correspond to a right turn of f. Note that an unconstrained maximal chain of f may consist of a single, degree one vertex.

We now construct two partially-directed graphs, one representing the "left" relation between maximal vertical chains of H, the other representing the "below" relation between maximal horizontal chains of H. The graph representing the "left" relation between maximal vertical chains of H is constructed as follows. We first augment H with the saturating edges of G_r and G_ℓ incident with an end-vertex of an unconstrained maximal vertical chain of H. We then orient the horizontal edges of H from left to right, reverse the orientation of the saturating edges of G_ℓ, and leave the vertical edges of H not oriented so that they can be traversed in both ways. We denote by H_x the resulting graph (see Fig. 5a). Similarly, the graph representing the "below" relation between maximal horizontal chains of H is constructed as follows. We first augment H with the saturating edges of G_r and G_ℓ incident with an end-vertex of an unconstrained maximal horizontal chain of H. We then orient the vertical edges of H from bottom to top and leave the horizontal edges of H not oriented so that they can be traversed in both ways. We denote by H_y the resulting graph (see Fig. 5b).

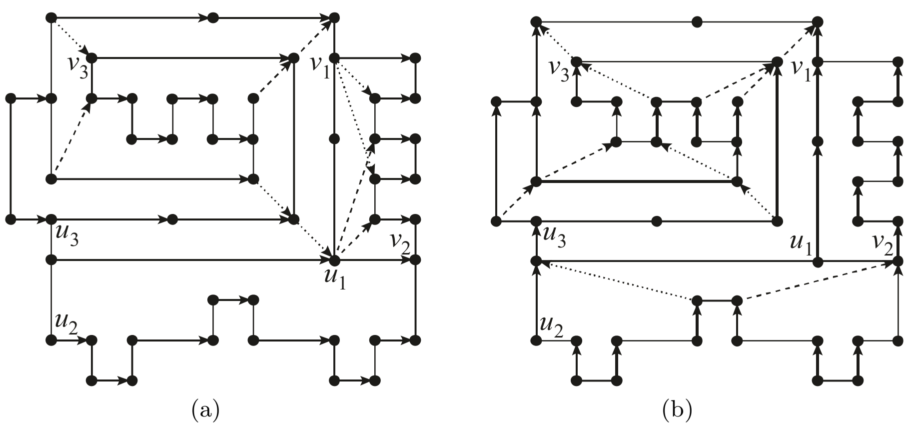


Fig. 5. (a) H_x. (b) H_y. Both graphs are obtained using the complete saturators shown in Fig. 4.

The following theorem shows how turn-regularity characterizes those orthogonal representations for which the "left" relation between maximal vertical chains and the "below" relation between maximal horizontal chains are uniquely determined.

Theorem 4. *Let H be an orthogonal representation of an embedded 4-planar graph. H_x and H_y are uniquely determined if and only if H is turn-regular.*

Note that H_x and H_y are no longer orthogonal representations, and may, in general, be non-planar. From the definition of saturator, it follows that each saturating edge from G_r and G_ℓ used in the construction of H_x and H_y has both end-vertices on the same face of H. Two saturating edges in H_x or H_y are said

to *cross* each other if their end-vertices appear alternately on the boundary of a common face of H. In the rest of the paper we refer to the maximal chains of non-oriented edges of H_x as maximal vertical chains of H_x, and denote by $mvc(v)$ the maximal vertical chain of H_x containing vertex v. Analogously, we refer to the maximal chains of non-oriented edges of H_y as maximal horizontal chains of H_y, and denote by $mhc(v)$ the maximal horizontal chain of H_y containing vertex v.

5 Turn-Regularity and Orthogonal Relations

In this section we use graphs H_x and H_y to characterize all possible orthogonal relations in a turn-regular orthogonal representation. This leads to a characterization of turn-regular orthogonal representations in terms of orthogonal relations. We denote by $u \rightarrow v$ a directed path from vertex u to vertex v in H_x containing at least a horizontal edge or in H_y containing at least a vertical edge, and by $u \not\rightarrow v$ the absence of such a path from vertex u to vertex v.

Lemma 2. *For each pair $\{u, v\}$ of vertices of H_x the following conditions hold:*

1. $mvc(u) = mvc(v)$ *if and only if* $u =_x v$
2. $u \rightarrow v$ *if and only if* $u <_x v$
3. $v \rightarrow u$ *if and only if* $v <_x u$
4. $mvc(u) \neq mvc(v)$, $u \not\rightarrow v$, *and* $v \not\rightarrow u$ *if and only if no x-relation can be established between u and v.*

As an example, we identify in the graph H_x shown in Fig. 5a three pairs of vertices corresponding to the various cases of Lemma 2: u_1 and v_1 belong to the same maximal vertical chain, there exists a directed path from u_2 to v_2, while there is neither a path between u_3 and v_3, nor they belong to the same maximal vertical chain.

Lemma 3. *For each pair $\{u, v\}$ of vertices of H_y the following conditions hold:*

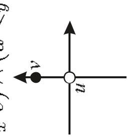
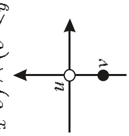
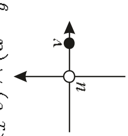
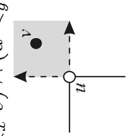
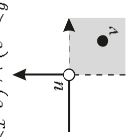
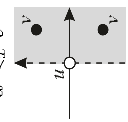
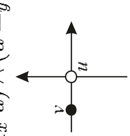
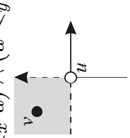
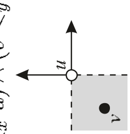
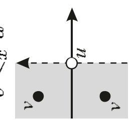
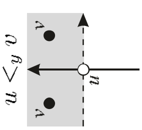
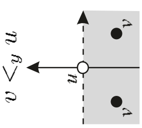
1. $mhc(u) = mhc(v)$ *if and only if* $u =_y v$
2. $u \rightarrow v$ *if and only if* $u <_y v$
3. $v \rightarrow u$ *if and only if* $v <_y u$
4. $mhc(u) \neq mhc(v)$, $u \not\rightarrow v$, *and* $v \not\rightarrow u$ *if and only if no y-relation can be established between u and v.*

Lemma 4. *Let G be an embedded 4-planar graph and H be a turn-regular orthogonal representation of G. For each pair $\{u, v\}$ of vertices of G, exactly one orthogonal relation holds (see Table 1).*

We are interested in those orthogonal representations for which there is an orthogonal relation between each pair of vertices. As the following theorem shows, this class of orthogonal representations is characterized by turn-regularity.

Theorem 5. *An orthogonal representation H of an embedded 4-planar graph is turn-regular if and only if there is an orthogonal relation between every two vertices of H.*

Table 1. Orthogonal relations for a pair $\{u, v\}$ of vertices in a turn-regular orthogonal representation H.

		H_y			
		$mhc(u) = mhc(v)$	$u \to v$	$v \to u$	otherwise
H_x	$mvc(u) = mvc(v)$	impossible	$(u =_x v) \wedge (u <_y v)$ 	$(u =_x v) \wedge (v <_y u)$ 	impossible
	$u \to v$	$(u <_x v) \wedge (u =_y v)$ 	$(u <_x v) \wedge (u <_y v)$ 	$(u <_x v) \wedge (v <_y u)$ 	$u <_x v$ 
	$v \to u$	$(v <_x u) \wedge (u =_y v)$ 	$(v <_x u) \wedge (u <_y v)$ 	$(v <_x u) \wedge (v <_y u)$ 	$v <_x u$ 
	otherwise	impossible	$u <_y v$ 	$v <_y u$ 	impossible

6 Turn-Regularity and Drawing Algorithms

In this section we first study the problem of efficiently checking whether an orthogonal representation is turn-regular. Then we show how an optimal area orthogonal drawing of a turn-regular orthogonal representation can be computed.

Theorem 6. *A turn-regular orthogonal representation of an embedded 4-planar graph with n vertices and bends can be recognized in $O(n)$ time and space.*

The optimal area drawings that we want to compute are planar. The next theorem guarantees the planarity of drawings that satisfy the orthogonal relations of a turn-regular orthogonal representation.

Theorem 7. *Let H be a turn-regular orthogonal representation of an embedded 4-planar graph, and let Γ be an orthogonal drawing of H such that, for each pair $\{u, v\}$ of vertices of H, the orthogonal relation between u and v is satisfied. Then Γ is planar.*

We are now ready to present two different algorithms that compute optimal area drawings of turn-regular orthogonal representations. These algorithms are variations of the two compaction procedures described in [6]. For the first algorithm, we define two digraphs, denoted D_x and D_y. D_x is obtained from H_x by shrinking each maximal vertical chain to a single vertex, by removing possible multiple edges, and by adding a super-source and a super-sink (see Fig 6a). Thus, there is a one-to-one correspondence between maximal vertical chains of H_x and vertices of D_x, and a many-to-one correspondence between directed edge of H_x and edges of D_x. Note that in the shrinking process we "preserve the embedding", i.e., the circular ordering of the edges around each vertex v of D_x is induced by the circular ordering of the directed edges "around" the maximal vertical chain of H_x corresponding to v. D_y is obtained analogously from H_y by shrinking the maximal horizontal chains (see Fig 6b).

Property 1. D_x and D_y are planar st-digraphs.

Theorem 8. *Let H be a turn-regular orthogonal representation of an embedded 4-planar graph, and let n be the number of vertices and bends of H. A planar orthogonal drawing of H with optimal area can be constructed in $O(n)$ time and space.*

Theorem 9. *Let H be a turn-regular orthogonal representation of an embedded 4-planar graph, and let n be the number of vertices and bends. A planar orthogonal drawing of H with optimal area $\mathcal{A}$ and whose total edge length is optimal among all drawings with area $\mathcal{A}$ can be constructed in $O(n^{7/4} \log n)$ time and $O(n)$ space.*

We recall that the minimum number of bends for an orthogonal representation of a 4-planar graph with n vertices is $O(n)$ [3, 29]. The algorithm described in [27] produces such an orthogonal representation, and there exist various algorithms for producing an orthogonal representation with a sub-optimal $O(n)$ number of bends (see, e.g, [4, 23, 28]).

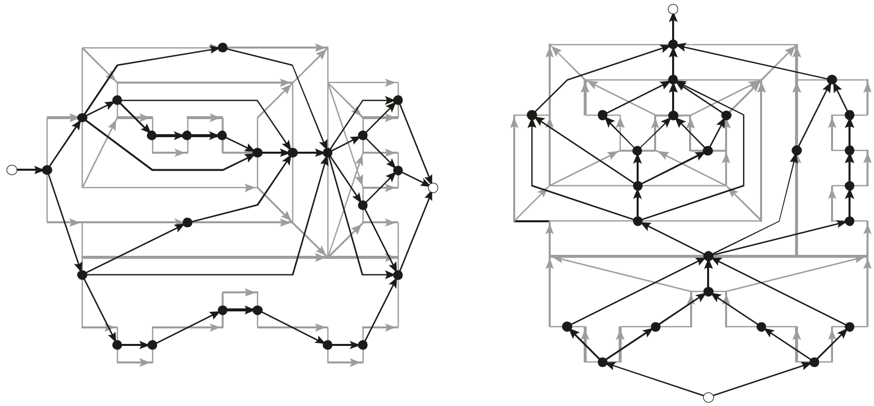


Fig. 6. (a) D_x. (b) D_y. The corresponding graphs H_x and H_y, from Fig. 5, are represented in grey.

7 Experiments

In this section, we present the results of an experimental study on a test suite of planar orthogonal representations of randomly generated biconnected 4-planar graphs. The analysis of the test suite has shown that the percentage of turn-regular faces is quite high. Motivated by this result, we have designed compaction heuristics based on the idea of "face turn-regularization".

7.1 Compaction Heuristics

We have implemented a compaction algorithm for orthogonal representations based on the results described in the previous sections. Namely, let H be a given orthogonal representation of an embedded 4-planar graph. The algorithm proceeds as follows:

– H is first tested for turn-regularity, using the algorithm described in Theorem 6.
– If H is turn-regular, the algorithm computes an orthogonal drawing of H with optimal area and optimal total edge length within that area by applying the techniques in Theorem 9.
– If H contains some faces that are not turn-regular, an algorithm is applied to make these faces turn-regular. The algorithm adds dummy vertices and edges to H, creating a new orthogonal representation H' that is turn-regular. The techniques in Theorem 9 are then used to find a drawing Γ' of H' with optimal area and optimal total edge length within that area. Finally, the dummy vertices and edges are removed from Γ' to yield an orthogonal drawing Γ of H. In general the orthogonal drawing Γ does not have optimal area and total edge length.
  Two simple approaches are used to make non-turn-regular faces turn-regular:

1. The first approach is an improvement of the standard rectangularization method described in [6, 27]. When a dummy edge is inserted, a dummy vertex is added only if it really needed. Each non-turn-regular face is divided into two or more smaller, rectangular faces.
2. The second approach recursively adds a straight edge (randomly chosen to be either horizontal or vertical) between each pair of kitty corners, until the face has been decomposed in smaller (but not necessarily rectangular) turn-regular faces. In general, this technique adds a much smaller number of dummy edges than the first approach.

In the following, we call the two heuristic compaction algorithms derived from the two turn-regularization approaches described above `Heur1` and `Heur2`, respectively. They are implemented in the *GDToolkit* library[4].

7.2 Test Suite and Experimental Results

Heuristics `Heur1` and `Heur2` were tested on a set of 530 randomly generated biconnected 4-planar graphs with number of vertices in the range $10 \ldots 3000$. The results are compared with a third compaction heuristic, `StdComp`, in which all faces, both turn-regular and not, are decomposed into rectangles using the rectangularization method of `Heur1`.

The graphs in the test suite have been generated with a technique used in other experimental studies on orthogonal drawings [1]. Each biconnected 4-planar graph is generated from a cycle of three vertices by performing a random series of *InsertVertex* and *InsertEdge* operations. The *InsertVertex* operation subdivides an existing edge into two new edges separated by a new vertex. The *InsertEdge* operation inserts a new edge between two existing vertices on the same face. Any biconnected planar graph can be generated by a sequence of these two operations. Also, for each graph to be generated, the density of the graph, i.e., the number of edges divided by the number of vertices, is randomly chosen before the generation algorithm is run.

Our first experiment consisted in studying the percentage of turn-regular faces in the graphs of our test suite. We have found that the percentage of turn-regular faces increases logarithmically with the density of the graphs, stabilizes at around 95%, and has an average value of 89% (see Fig. 7).

We have then analyzed the results of the three heuristics. In particular, we have considered, for `Heur1` and `Heur2`, the improvement in the drawing area, total edge length, and maximum edge length with respect to `StdComp`. `Heur2` performs better than `Heur1` in most cases; also, the improvement in area and total edge length of `Heur2` with respect to `StdComp` increases with the number of vertices of the graph (see Fig. 8). The average improvements of area and total edge length are 25% and 19%, respectively, for graphs with 3000 vertices; and there are some graphs in the test suite for which the area improvement is more than 45%.

[4] `http://www.dia.uniroma3.it/~gdt/`

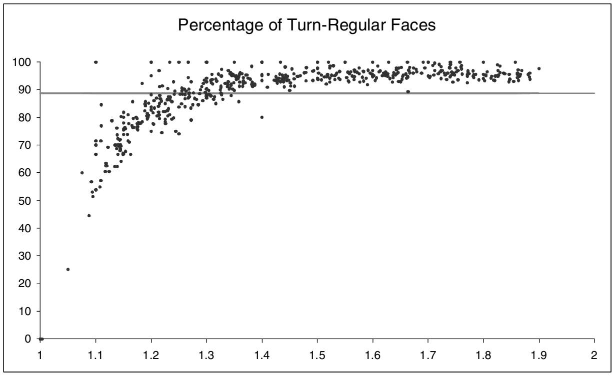


Fig. 7. The percentage of turn-regular faces of each graph. The x-axis indicates the density. The horizontal line indicates the average value.

We have also executed the three heuristics on a very large graph with $10,000$ vertices. The drawing computed by `Heur2` improves the area by 41% and the total edge length by 22% with respect to the drawing computed by `StdComp`.

8 Conclusions and Future Work

We introduced the notion of turn-regularity which allows the characterization of a class of orthogonal representations that are optimally compactable in terms of area in polynomial-time. In particular, given a turn-regular orthogonal representation of an embedded 4-planar graph, we provided a linear-time algorithm to compute a planar drawing with minimum area, and a polynomial-time algorithm to compute a planar drawing with optimal area and minimum total edge length within that area.

We provided several implementations of heuristics for making orthogonal representations turn-regular and we used them in the compaction algorithm, in place of the standard rectangularization step. Experiments on a randomly generated test suite of biconnected 4-planar graphs showed that the new compaction strategy performs much better than the standard one, especially for very large graphs.

The results presented in this paper motivate some future work, which includes:

- To continue the experimental study of the described heuristics on non-biconnected 4-planar graphs, comparing their behavior also with that of VLSI compaction algorithms.
- To investigate other effective heuristics for making an orthogonal representation turn-regular by adding a small number of edges.

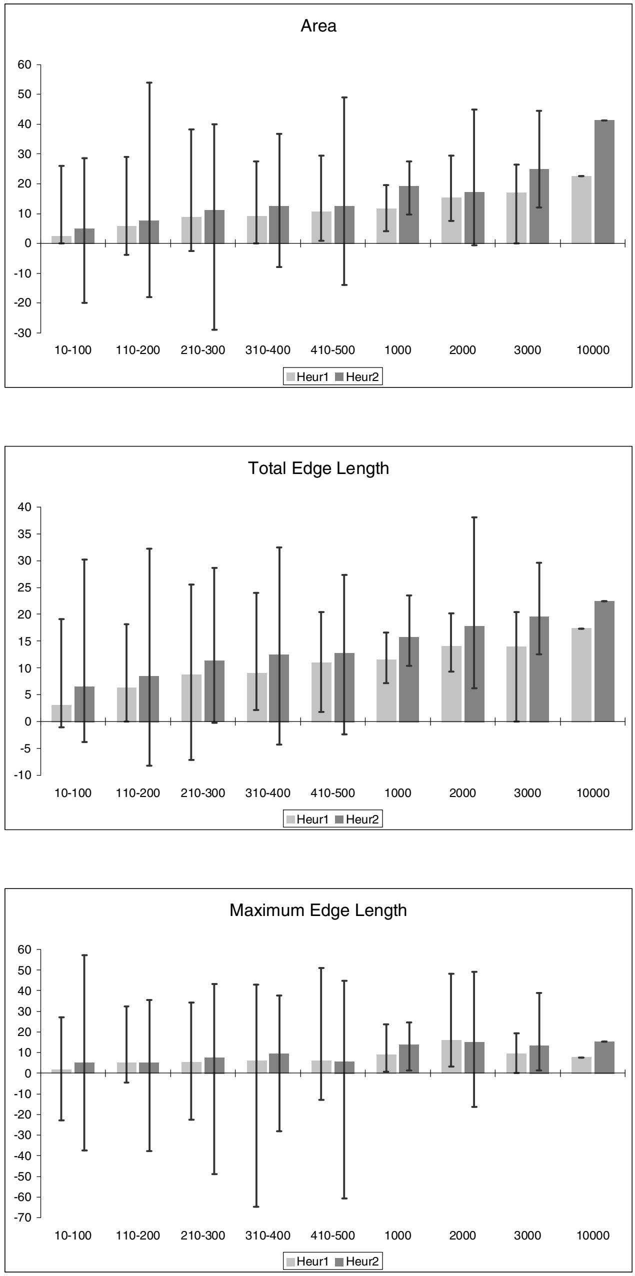


Fig. 8. The average percentage improvement in area, total edge length, and maximum edge length of `Heur1` and `Heur2` with respect to `StdComp`. The x-axes indicate the number of vertices. The interval lines indicate the minimum and maximum improvement.

– To find other families of orthogonal representations for which an optimal area drawing can be computed in polynomial time.
– The problem of computing an orthogonal representation with the minimum number of bends has been extensively investigated in a variable embedding setting [9, 11, 13]. It would be interesting to study the compaction problem when it is possible to change the embedding of the input graph.

Acknowledgments

We would like to thank Maurizio Patrignani for useful discussions.

References

[1] P. Bertolazzi, G. Di Battista, and W. Didimo. Computing orthogonal drawings with the minimum number of bends. In F. Dehne, A. Rau-Chaplin, J.-R. Sack, and R. Tamassia, editors, *Algorithms and Data Structures (Proc. WADS '97)*, volume 1272 of *Lecture Notes Comput. Sci.*, pages 331–344. Springer-Verlag, 1997.
[2] P. Bertolazzi, G. Di Battista, G. Liotta, and C. Mannino. Upward drawings of triconnected digraphs. *Algorithmica*, 6(12):476–497, 1994.
[3] T. C. Biedl. New lower bounds for orthogonal graph drawings. In F. J. Brandenburg, editor, *Graph Drawing (Proc. GD '95)*, volume 1027 of *Lecture Notes Comput. Sci.*, pages 28–39. Springer-Verlag, 1996.
[4] T. C. Biedl and G. Kant. A better heuristic for orthogonal graph drawings. *Comput. Geom. Theory Appl.*, 9(3):159–180, 1998.
[5] J. A. Bondy and U. S. R. Murty. *Graph Theory with Applications*. North-Holland, Amsterdam, The Netherlands, 1976.
[6] G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. *Graph Drawing*. Prentice Hall, Upper Saddle River, NJ, 1999.
[7] G. Di Battista, A. Garg, G. Liotta, R. Tamassia, E. Tassinari, and F. Vargiu. An experimental comparison of four graph drawing algorithms. *Comput. Geom. Theory Appl.*, 7(5–6):303–325, 1997.
[8] G. Di Battista and G. Liotta. Upward planarity checking: "Faces are more than polygons". In S. H. Whitesides, editor, *Graph Drawing (Proc. GD '98)*, volume 1547 of *Lecture Notes Comput. Sci.*, pages 72–86. Springer-Verlag, 1998.
[9] G. Di Battista, G. Liotta, and F. Vargiu. Spirality and optimal orthogonal drawings. *SIAM J. Comput.*, 27(6):1764–1811, 1998.
[10] G. Di Battista and R. Tamassia. Algorithms for plane representations of acyclic digraphs. *Theoret. Comput. Sci.*, 61(2,3):175–198, 1988.
[11] W. Didimo and G. Liotta. Computing orthogonal drawings in a variable embedding setting. In K.-Y. Chwa and O. H. Ibarra, editors, *Algorithms and Computation (Proc. ISAAC '98)*, volume 1533 of *Lecture Notes Comput. Sci.*, pages 79–88. Springer-Verlag, 1998.
[12] U. Fößmeier and M. Kaufmann. Drawing high degree graphs with low bend numbers. In F. J. Brandenburg, editor, *Graph Drawing (Proc. GD '95)*, volume 1027 of *Lecture Notes Comput. Sci.*, pages 254–266. Springer-Verlag, 1996.
[13] A. Garg and R. Tamassia. Planar drawings and angular resolution: Algorithms and bounds. In J. van Leeuwen, editor, *Algorithms (Proc. ESA '94)*, volume 855 of *Lecture Notes Comput. Sci.*, pages 12–23. Springer-Verlag, 1994.

[14] A. Garg and R. Tamassia. On the computational complexity of upward and rectilinear planarity testing. In R. Tamassia and I. G. Tollis, editors, *Graph Drawing (Proc. GD '94)*, volume 894 of *Lecture Notes Comput. Sci.*, pages 286–297. Springer-Verlag, 1995.

[15] A. Garg and R. Tamassia. A new minimum cost flow algorithm with applications to graph drawing. In S. North, editor, *Graph Drawing (Proc. GD '96)*, volume 1190 of *Lecture Notes Comput. Sci.*, pages 201–216. Springer-Verlag, 1997.

[16] N. Gelfand and R. Tamassia. Algorithmic patterns for orthogonal graph drawing. In S. H. Whitesides, editor, *Graph Drawing (Proc. GD '98)*, volume 1547 of *Lecture Notes Comput. Sci.*, pages 138–152. Springer-Verlag, 1998.

[17] F. Harary. *Graph Theory*. Addison-Wesley, Reading, MA, 1969.

[18] F. Hoffmann and K. Kriegel. Embedding rectilinear graphs in linear time. *Inform. Process. Lett.*, 29(2):75–79, 1988.

[19] M. Y. Hsueh and D. O. Pederson. Computer-aided layout of lsi circuit building-blocks. In *Proc. IEEE Internat. Symp. Circuits and Systems*, 1979.

[20] G. W. Klau and P. Mutzel. Optimal compaction of orthogonal grid drawings. In G. Cornuejols, R. E. Burkard, and G. J. Woeginger, editors, *Integer Programming and Combinatorial Optimization (Proc. IPCO '99)*, volume 1610 of *Lecture Notes Comput. Sci.*, pages 304–319. Springer-Verlag, 1999.

[21] T. Lengauer. *Combinatorial Algorithms for Integrated Circuit Layout.* B. G. Teubner – John Wiley & Sons, Stuttgart, Germany – Chichester, England, 1990.

[22] R. H. J. M. Otten and J. G. van Wijk. Graph representations in interactive layout design. In *Proc. IEEE Internat. Sympos. Circuits and Systems*, pages 914–918, 1978.

[23] A. Papakostas and I. G. Tollis. Algorithms for area-efficient orthogonal drawings. *Comput. Geom. Theory Appl.*, 9(1–2):83–110, 1998. Special Issue on Geometric Representations of Graphs, G. Di Battista and R. Tamassia, editors.

[24] M. Patrignani. On the complexity of orthogonal compaction. In F. Dehne, A. Gupta, J.-R. Sack, and R. Tamassia, editors, *Algorithms and Data Structures (Proc. WADS '99)*, volume 1663 of *Lecture Notes Comput. Sci.*, pages 56–61. Springer-Verlag, 1999.

[25] J. M. Six, K. G. Kakoulis, and I. G. Tollis. Refinement of orthogonal graph drawings. In S. H. Whitesides, editor, *Graph Drawing (Proc. GD '98)*, volume 1547 of *Lecture Notes Comput. Sci.*, pages 302–315. Springer-Verlag, 1998.

[26] L. Stockmeyer. Optimal orientation of cells in slicing floorplan design. *Inform. Control*, 57(2/3):91–101, 1983.

[27] R. Tamassia. On embedding a graph in the grid with the minimum number of bends. *SIAM J. Comput.*, 16(3):421–444, 1987.

[28] R. Tamassia and I. G. Tollis. Planar grid embedding in linear time. *IEEE Trans. Circuits Syst.*, CAS-36(9):1230–1234, 1989.

[29] R. Tamassia, I. G. Tollis, and J. S. Vitter. Lower bounds for planar orthogonal drawings of graphs. *Inform. Process. Lett.*, 39(1):35–40, 1991.

[30] G. Vijayan and A. Wigderson. Rectilinear graphs and their embeddings. *SIAM J. Comput.*, 14(2):355–372, 1985.

[31] W. Wimer, I. Koren, and I. Cederbaum. Floorplans, planar graphs and layouts. *IEEE Trans. Circuits Syst.*, CAS-35(3):267–278, 1988.

Combining Graph Labeling and Compaction*
(Extended Abstract)

Gunnar W. Klau and Petra Mutzel

Max–Planck–Institut für Informatik
Im Stadtwald, D–66123 Saarbrücken, Germany
{guwek,mutzel}@mpi-sb.mpg.de

Abstract. Combinations of graph drawing and map labeling problems yield challenging mathematical problems and have direct applications, *e.g.*, in automation engineering. We call graph drawing problems in which subsets of vertices and edges need to be labeled *graph labeling problems*. Unlike in map labeling where the position of the objects is specified in the input, the coordinates of vertices and edges in a graph labeling problem instance have yet to be determined and thus create additional degrees of freedom. We concentrate on the *Compaction and Labeling (COLA) Problem*: Given an orthogonal representation—as produced by algorithms within the topology–shape–metrics paradigm—and some label information, the task is to generate a labeled orthogonal embedding with minimum total edge length. We characterize feasible solutions of the *COLA* problem extending an existing framework for solving pure compaction problems. Based on the graph–theoretical characterization, we present a branch–and–cut algorithm which computes optimally labeled orthogonal drawings for given instances of the *COLA* problem. First computational experiments on a benchmark set of practical instances show that our method is superior to the traditional approach of applying map labeling algorithms to graph drawings. To our knowledge, this is the first algorithm especially designed to solve graph labeling problems.

1 Introduction

The area of graph drawing provides an ample variety of algorithms that produce aesthetically pleasing layouts of graphs [DETT99]. At the same time, there are sophisticated techniques to generate provably good labelings of cartographic maps [WS]. Combinations of problems from both areas yield challenging mathematical problems and have direct applications, e.g., the automatic layout of state diagrams in automation engineering or the drawing of schematic maps such as subway maps. Unlike in map labeling where the position of the objects is specified in the input, the coordinates of vertices and edges in a graph drawing problem instance have yet to be determined and thus create additional degrees of freedom.

* This work is partially supported by the Bundesministerium für Bildung, Wissenschaft, Forschung und Technologie (No. 03–MU7MP1–4).

J. Kratochvíl (Ed.): GD'99, LNCS 1731, pp. 27–37, 1999.

We call the class of graph drawing problems where subsets of vertices and edges have to be labeled *graph labeling problems.* So far, problems from this class have only been attacked by applying map labeling algorithms to graph drawings [KT97,IL97]. Figure 1 shows a labeled drawing for a state diagram; first the underlying graph has been drawn, in a last step the labels have been placed with an optimal labeling strategy. This forces some labels to either overlap parts of the drawing or to be placed far away from the vertices to which they should be attached. Another approach could be to model each label as a vertex of fixed size and then apply a graph drawing algorithm which can cope with vertices of different sizes. This treatment increases the vertex degrees and may destroy existing structural properties in the graph—for the drawing of state diagrams this leads to a loss of orthogonality and unacceptable area consumption.

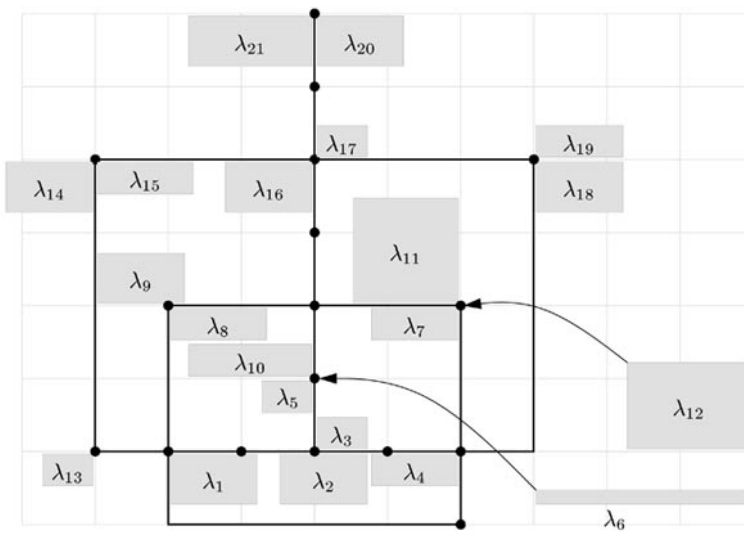


Fig. 1. A state diagram; first drawn then labeled

In this paper we concentrate on a special graph labeling problem: the *Compaction and Labeling (COLA) Problem.* Graph drawing algorithms in the *topology–shape–metrics paradigm* [DETT99] produce in an intermediate step a so–called *orthogonal representation* H, i.e., an extended planar representation. In addition to the topological information, H contains for each edge information about the order of bends and the angle formed with the following edge in the appropriate face.

In the *compaction* phase of the paradigm, the goal is to minimize the area or the total edge length of the resulting orthogonal drawing. The *COLA* problem unifies the problem to find a drawing respecting the given shape and an additional labeling task: A set of labels, each of which is fixed in size, is assigned to every vertex v. Each of the labels must be placed so that it touches v in at

least one point and does not overlap other objects. We are given a set of k labels $L = \{\lambda_1, \lambda_2, \ldots, \lambda_k\}$, two functions $w : L \to \mathbb{Q}$, $h : L \to \mathbb{Q}$ determining the width and height of each label, and a third function $a : L \to V$ denoting the vertex $a(\lambda)$ to which the label λ should be attached. We are looking for a labeled orthogonal embedding with short edges in which each label is placed correctly, and no labels overlap the drawing.

In this paper we give a characterization of feasible solutions for the *COLA* problem. We extend an existing framework for two–dimensional compaction problems in graph drawing [KM99]. Based on the graph–theoretical characterization, we present a first branch–and–cut based algorithm prototype which computes optimally labeled orthogonal drawings for given instances of the *COLA* problem. To our knowledge, this is the first algorithm especially designed to solve graph labeling problems.

Section 2 introduces the graph–theoretical framework. We give a formal definition of the *COLA* problem in Sect. 3 and show how to integrate the additional labeling problem into the framework. In Sect. 4 we extend an optimal branch–and–cut algorithm so that it solves the *COLA* problem to optimality and give some first computational results. Finally, we conclude with Sect. 5. Due to space limitations we can only give sketches of the proofs.

2 Constraint Graphs

In this section we introduce the graph–theoretical framework we have developed to solve the *COLA* problem. An algorithm within the topology–shape–metrics approach [DETT99] produces at the end of the second phase an orthogonal representation H which contains the information about the topology and the shape of the drawing. We call H *simple* if its number of bends is zero. Every orthogonal representation can be transformed into a simple one by just replacing each bend with an artificial vertex.

In order to compute a drawing for H, we must assign coordinates to the vertices and bends. In this paper, we concentrate on *pure orthogonal embeddings*, but our ideas can be adapted to other drawing standards such as *Kandinsky–like embeddings* (introduced in [FK96]) or *orthogonal box embeddings* (subclasses are the *big node model* from [FK97] and the *TSS* model from [BMT97], a related class is the *quasi–orthogonal model* from [KM98]).

Pure orthogonal embeddings are only admissible for 4–planar graphs. Each vertex and bend is mapped to a point and edge segments are mapped to horizontal or vertical non–crossing line segments of some minimum length connecting the images of their endpoints. A special case are pure orthogonal grid embeddings as defined in [Tam87]: Here, vertices and bends have integer coordinates. As with representations, we call orthogonal embeddings simple if they do not contain bends.

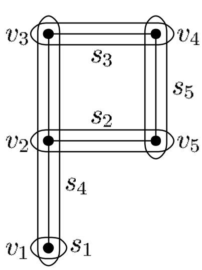
We first focus on the following subproblem of the *COLA* problem: Given an orthogonal representation H, produce an orthogonal embedding Γ for H

which is—among all such drawings—of minimum total edge length. Recently, Patrignani proved the NP–hardness of this problem [Pat99].

Initial observation leads to the following notion: Incident edges of same direction can be combined, forming the objects to compact. Let Γ be a simple orthogonal drawing of a graph. It induces a partition of the set of edges E into the horizontal set E_h and the vertical set E_v. A horizontal (resp. vertical) *subsegment* in Γ is a connected component in (V, E_h) (resp. (V, E_v)). If the component is maximally connected it is also referred to as a *segment*.

The following observations are crucial (see also Fig. 2).

- Each edge is a subsegment.
- Each vertex v belongs to one unique horizontal and one unique vertical segment, denoted by $\mathrm{hor}(v)$ and $\mathrm{vert}(v)$.
- The *limits* of a subsegment s are given as follows: Let v_l, v_r, v_b, and v_t be the leftmost, rightmost, bottommost, and topmost vertices on s. Then $l(s) = \mathrm{vert}(v_l)$, $r(s) = \mathrm{vert}(v_r)$, $b(s) = \mathrm{hor}(v_b)$, and $t(s) = \mathrm{hor}(v_t)$.



$S_h = \{s_1, s_2, s_3\}$ $\quad$ $S_v = \{s_4, s_5\}$

i	$V(s_i)$	$E(s_i)$	$l(s_i)$	$r(s_i)$	$b(s_i)$	$t(s_i)$
1	$\{v_1\}$	$\{\}$	s_4	s_4	s_1	s_1
2	$\{v_2, v_5\}$	$\{(v_2, v_5)\}$	s_4	s_5	s_2	s_2
3	$\{v_3, v_4\}$	$\{(v_3, v_4)\}$	s_4	s_5	s_3	s_3
4	$\{v_1, v_2, v_3\}$	$\{(v_1, v_2), (v_2, v_3)\}$	s_4	s_4	s_1	s_3
5	$\{v_4, v_5\}$	$\{(v_4, v_5)\}$	s_5	s_5	s_2	s_3

Fig. 2. Segments of a simple orthogonal grid drawing and its limits

We provide a necessary and sufficient condition for all feasible solutions of a given instance of the compaction problem. This condition is based on existing paths in the so–called *constraint graphs*. This pair of directed graphs is similar to the *layout graphs* known from one–dimensional compaction strategies in VLSI design [Len90]. Nodes in these graphs represent the segments; their weighted arcs characterize relative positioning relations between the segments. We denote by c_i the coordinate of segment s_i. A weight $\omega_{ij} \in \mathbb{Q}$ for an arc (s_i, s_j) indicates that the coordinate difference $c_j - c_i$ must be at least ω_{ij}. Figure 3 shows an example for a pair of constraint graphs. The arcs specify exactly the relative relationships known from the shape of the graph: We call such special pairs of constraint graphs $\langle (S_v, A_h), (S_h, A_v) \rangle$ *shape graphs* or a *shape description.* Note that each horizontal edge in the original graph defines a relative positioning constraint between two vertical segments. Similarly, vertical edges determine constraints between horizontal segments.

The characterization of feasible solutions for the two-dimensional compaction problem is based on the following three observations:

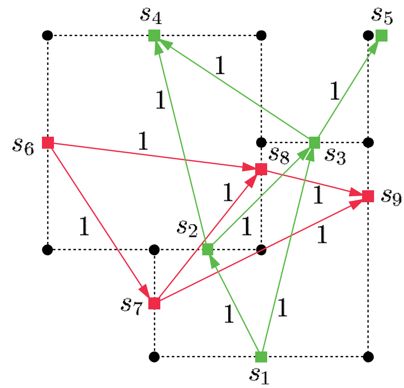


Fig. 3. An orthogonal grid embedding (dotted) and a pair of constraint graphs

1. The arcs of the shape graphs are contained in every pair of constraint graphs corresponding to a drawing reflecting the given shape.
2. Generally, the information in the shape graphs is not sufficient to produce an orthogonal embedding. If this is the case, however, we call such a pair of constraint graphs *complete*.
3. There are in general many possibilities for extending the shape graphs to a complete pair of constraint graphs.

We denote by $s_i \xrightarrow{+} s_j$ a path of non–negative weight between s_i and s_j. The following is a precise characterization for complete pairs of constraint graphs in terms of paths that must be contained in the arc sets: A pair of graphs is complete if and only if both arc sets do not contain non–negative cycles and for every pair of segments $(s_i, s_j) \in S \times S$ one of the following four conditions holds:

$$\begin{array}{llll} 1. & r(s_i) \xrightarrow{+} l(s_j) & 3. & t(s_j) \xrightarrow{+} b(s_i) \\ 2. & r(s_j) \xrightarrow{+} l(s_i) & 4. & t(s_i) \xrightarrow{+} b(s_j) \end{array} \tag{1}$$

If one of the non–negative paths in (1) exists we also call the pair of segments (s_i, s_j) *separated*. Note that this definition generalizes the notion of completeness as introduced for the pure compaction problem in [KM99]. There, we have the special case that $\omega_{ij} = 1$ for all (s_i, s_j).

We can now express a one–to–one correspondence between these complete extensions and orthogonal embeddings. On the basis of the following theorem, the two–dimensional compaction task can be seen as the search for a complete extension of the given shape graphs leading to minimum total edge length.

Theorem 1 ([KM99]). *For each simple orthogonal embedding with shape description* $\sigma = \langle (S_v, A_h), (S_h, A_v) \rangle$ *there exists a complete extension* $\tau = \langle (S_v, B_h), (S_h, B_v) \rangle$ *of* σ *and vice versa: Every complete extension* $\tau = \langle (S_v, B_h),$

$(S_h, B_v)\rangle$ of a shape description $\sigma = \langle (S_v, A_h), (S_h, A_v)\rangle$ corresponds to a simple orthogonal embedding with shape description σ.

3 Modeling the Labels

In this section we extend the graph–theoretical formulation of the pure two–dimensional compaction problem so that a feasible solution corresponds to a labeled orthogonal embedding and thus to a solution of the *COLA* problem. We define the characteristics of a solution for the *COLA* problem in order to state the *COLA* problem formally.

Definition 1. *A* labeled orthogonal embedding *Γ_L for an orthogonal representation H of a planar graph $G = (V, E)$ and a label set L with functions $w, h : L \to \mathbb{Q}$ and $a : L \to V$ fulfills the following properties:*

1. *Γ_L is an orthogonal embedding for H.*
2. *A label $\lambda \in L$ has size $w(\lambda) \times h(\lambda)$ and its image intersects with the image of vertex $a(\lambda)$ at at least one point.*
3. *A label $\lambda \in L$ does neither overlap nor include other objects.*

Definition 2. Compaction and Labeling (COLA) problem.
Given an orthogonal representation H for a planar graph $G = (V, E)$, a label set L with functions $w, h : L \to \mathbb{Q}$, and $a : L \to V$, find a labeled orthogonal embedding for H with minimum total edge length.

We model each label λ as a rectangle bounded by the segments l_λ, r_λ, t_λ, and b_λ as in Fig. 4. Its limits are linked by two arcs $(l_\lambda, r_\lambda) \in A_h$ and $(b_\lambda, t_\lambda) \in A_v$ of weight $w(\lambda)$ and $h(\lambda)$, respectively. Unlike the arcs from the previous section, the minimum distance requirements for this type of arcs are determined by the width and height of the corresponding label. Observe that we use the same sort of segments we have introduced in the previous section for the solution of the compaction problem. At this point we introduce a set O containing the objects to compact. It consists of the segments corresponding to consecutive edges of same direction and all the labels. This gives us an important property free of charge: If we can ensure that—as in Sect. 2—all pairs of distinct objects in O are separated, the labels will neither overlap nor will they be crossed by edges of the graph. We will achieve this by giving a more general definition of completeness.

We additionally have to guarantee that each label is drawn in the neighborhood of the vertex to which it belongs. Figure 5 shows the possibilities we want to capture with our formulation. According to Def. 1 a label can be placed anywhere close to a vertex, i.e., it must touch the vertex at at least one point.

The relative positioning of label λ to its vertex $v = a(\lambda)$ can be equivalently expressed by the following four conditions:

1. The left side of λ must not lie to the right side of v.
2. The right side of λ must not lie to the left side of v.

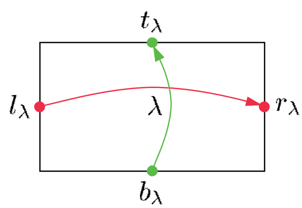


Fig. 4. Four segments model a label

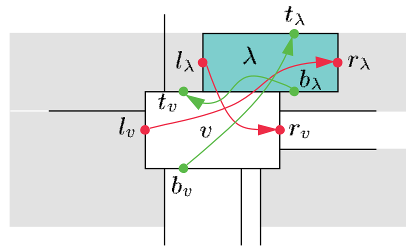


Fig. 5. Possibilities for placing a label λ in the neighborhood of $a(\lambda)$

3. The bottom side of λ must not lie above the top side of v.
4. The top side of λ must not lie below the bottom side of v.

In our graph–theoretical framework this translates into four special arcs of zero weight that have to be included in the pair of constraint graphs (see Fig. 5). For a label λ with $a(\lambda) = v$ these are (l_λ, r_v), (l_v, r_λ), (b_λ, t_v), and (b_v, t_λ), linking the limits of the label to the limits of the vertex. Note that we model vertices whose images are boxes of non–zero size in a similar way as the labels by including them in O and linking them appropriately to the segments that correspond to incident edges.

Having introduced these new types of arcs, we can extend the notion of *shape description* as defined in Sect. 2 to integrate the labels in the framework.

Definition 3. *A* labeled shape description σ_L *of an orthogonal representation* H *with label information* L, w, h, *and* a *is a tuple* $\langle (S_v \cup S_{L_v}, A_h \cup A_{L_h}), (S_h \cup S_{L_h}, A_v \cup A_{L_v}) \rangle$ *where* $\langle (S_v, A_h), (S_h, A_v) \rangle$ *is a shape description for* H *and*

$$S_{L_v} = \bigcup_{\lambda \in L} \{l_\lambda, r_\lambda\}, \qquad A_{L_h} = \bigcup_{\lambda \in L} \{(l_\lambda, r_\lambda), (l_\lambda, r_{a(\lambda)}), (l_{a(\lambda)}, r_\lambda)\},$$

$$S_{L_h} = \bigcup_{\lambda \in L} \{b_\lambda, t_\lambda\}, \qquad A_{L_v} = \bigcup_{\lambda \in L} \{(b_\lambda, t_\lambda), (b_\lambda, t_{a(\lambda)}), (b_{a(\lambda)}, t_\lambda)\} \ .$$

Clearly, each instance of the *COLA* problem uniquely determines a labeled shape description. We can now give a more general formulation of completeness by considering all objects in the set O: A pair of labeled constraint graphs is complete if and only if it does not contain a non–negative cycle and each distinct pair of objects is separated. This generalization leads to the following main result.

Theorem 2. *For each simple labeled orthogonal embedding with shape description* $\sigma = \langle (S_v, A_h), (S_h, A_v) \rangle$ *and label information* L, w, h, a *there exists a complete labeled extension* $\tau = \langle (S_v \cup S_{L_v}, A_h \cup A_{L_h}), (S_h \cup S_{L_h}, A_v \cup A_{L_h}) \rangle$ *of* σ *and vice versa: Every complete labeled extension* $\tau = \langle (S_v \cup S_{L_v}, B_h \cup A_{L_h}), (S_h \cup S_{L_h}, B_v \cup A_{L_h}) \rangle$ *of a labeled shape description* $\sigma = \langle (S_v \cup S_{L_v}, A_h \cup A_{L_h}), (S_h \cup S_{L_h}, A_v \cup A_{L_h}) \rangle$ *corresponds to a simple labeled orthogonal embedding with shape description* σ*.*

Proof (Sketch). The proof is similar to the one in [KM99, Theorem 1], so we give only a sketch of it: For the forward direction, we construct a complete extension of the given labeled shape description σ by inserting the missing information according to the visibility properties in the drawing. Clearly, this excludes non–negative cycles. The separation properties are also fulfilled for all segments because otherwise there would be overlapping objects in the drawing, which is not the case.

For the other direction, we give a constructive proof of how to build a labeled orthogonal embedding with a given complete extension of a labeled shape description: As in the proof for the pure compaction setting, we make use of a *length assignment* to assign the coordinates to the segments—and thus to the vertices, bends, and labels. Again, we are able to show that the resulting drawing is a labeled orthogonal embedding reflecting the shape of H and the labeling information given in L, w, h, and a.

4 Optimal Graph Labeling

In this section we introduce our the ILP formulation to solve the *COLA* problem to optimality and present first computational results.

One way to characterize the set of complete extensions is by means of an integer linear program. We introduce a binary variable x_{ij} for each arc (s_i, s_j) that might be in some extension of the given labeled shape description $\sigma_L = \langle (S_v, A_h), (S_h, A_v) \rangle$. If (s_i, s_j) is contained in the extension, the corresponding variable x_{ij} is one, otherwise zero. We refer by X to the set of binary variables. Additionally, there is a variable $c_s \in \mathbb{Q}$ for each segment $s \in S$ denoting the coordinate of s. The ILP then reads as follows:

$$\min \sum_{e \in E_h} c_{r(e)} - c_{l(e)} + \sum_{e \in E_v} c_{t(e)} - c_{b(e)} \quad \text{subject to} \tag{2}$$

$$x_{r_o,l_p} + x_{r_p,l_o} + x_{t_o,b_p} + x_{t_p,b_o} \geq 1 \qquad \forall (o,p) \in O \times O, o \neq p \tag{2.1}$$

$$c_j - c_i \geq \omega_{ij} \qquad \forall (s_i, s_j) \in A_h \cup A_v \tag{2.2}$$

$$c_j - c_i - (M + \omega_{ij}) x_{ij} \geq -M \qquad \forall x_{ij} \in X \tag{2.3}$$

$$x_{ij} \in \{0, 1\} \qquad \forall x_{ij} \in X \tag{2.4}$$

Inequalities (2.1) model the characterization of separation, i.e., the existence of necessary paths between distinct objects in an extension. Inequalities (2.2) force the coordinates to obey the distance rules coded by the weighted arcs in the underlying labeled shape description; the value ω_{ij} denotes the minimum distance between s_i and s_j. The same must hold true for the potential additional arcs: Whenever a variable x_{ij} is one, we want an inequality of type (2.2), otherwise there should be no restriction on the coordinate variables. This situation is modeled by inequalities (2.3) with the help of a big constant M. We have shown in [KM99] that inequalities (2.2) and (2.3) ensure the absence of non–negative cycles in the extension.

Note that we do not introduce additional decision variables to integrate the labeling task; the choice of where to place a label is performed by the separation properties. A label can be placed wherever it does not interfere with other objects as long as it stays in the neighborhood of the vertex to which it belongs. Moreover, we do not restrict the label placement to a finite number of prescribed places. Of course, the number of possibly non–separated segments is much higher than in an instance of a pure compaction problem.

Like the one–to–one correspondence between complete extensions and orthogonal embeddings there is a one–to–one correspondence between feasible solutions of the ILP and complete extensions of the given labeled shape graphs:

Theorem 3. *For each feasible solution* (x, c) *of the ILP for a given labeled shape description* σ_L *there is a labeled orthogonal embedding with appropriate shape and label information and vice versa.*

Proof (Sketch). A solution of the ILP corresponds to a complete extension of the labeled shape description σ with appropriate length assignment. The forward direction follows with Theorem 1. Conversely, we can use the information of a labeled embedding to construct a feasible solution of the ILP. We can show that none of the inequalities is violated and that the value of the objective function equals the total edge length.

We have extended the existing branch–and–cut framework for the pure compaction problem to solve instances of the *COLA* problem and have developed a prototype of an optimal compaction and labeling algorithm. We have tested the algorithm with real–world instances from the area of automation engineering and our first computational experiments indicate that we are able to draw and label the state diagrams within moderate computation time. Figure 6 shows the output of the optimal algorithm for the example of Fig. 1. It took the algorithm prototype 29 seconds to compute the optimal solution.

5 Conclusions and Future Plans

We have introduced a combined compaction and labeling problem arising from a practical application in automation engineering. The task is to simultaneously assign consistent edge lengths for a given orthogonal representation which determines the shape of the drawing and to label subsets of vertices. The resulting drawing should have small total edge length and all labels should be placed at the corresponding vertices so that they do not overlap other objects. Already the pure compaction task is NP–hard.

We have integrated the labeling problem into an existing graph–theoretical framework for solving two–dimensional compaction problems and have introduced the notion of labeled orthogonal embedding. Inside this framework, we have devised a branch–and–cut algorithm that produces optimal solutions for the *COLA* problem. Note that we do not require a label to be placed on a finite

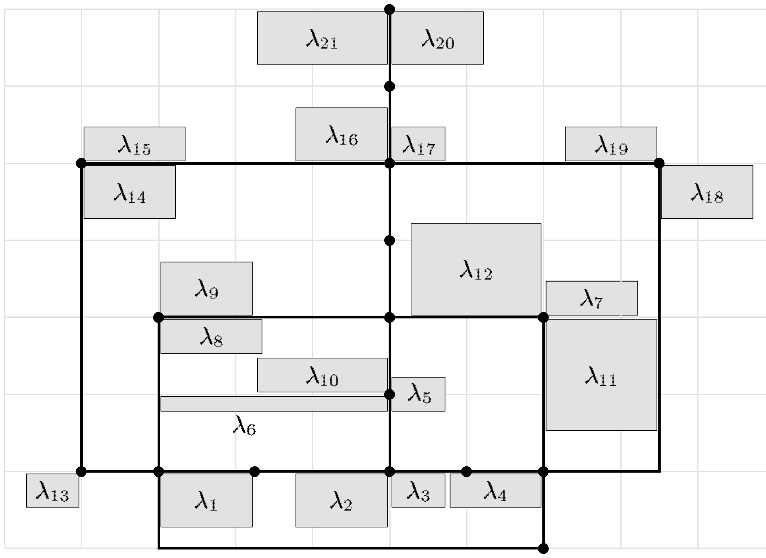


Fig. 6. The example from Fig. 1, drawn with the optimal algorithm

set of prescribed places but allow for a continuous movement of the label around the vertex to which it is attached.

In the near future, we will provide the implementation of the branch–and–cut algorithm as a module inside the AGD library [AGD99]. We will extensively test the optimal algorithm on real world and randomly generated instances. We will try to develop heuristics in order to apply them at each node in the branch–and–cut tree and want to investigate a possible extension to edge labeling problems. Furthermore, we plan to extend our framework so that it can accomplish some postprocessing tasks like improving the number of bends or crossings and decreasing the sizes of vertices.

References

AGD99. AGD. *AGD User Manual.* Max-Planck-Institut Saarbrücken, Universität Halle, Universität Köln, 1999. `http://www.mpi-sb.mpg.de/AGD`.

BMT97. T. Biedl, B. Madden, and I. Tollis. The three–phase method: A unified approach to orthogonal graph drawing. In G. Di Battista, editor, *Graph Drawing (Proc. GD '97)*, volume 1353 of *Lecture Notes in Computer Science*, pages 391–402. Springer-Verlag, 1997.

DETT99. G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. *Graph Drawing.* Prentice Hall, 1999.

FK96. U. Fößmeier and M. Kaufmann. Drawing high degree graphs with low bend numbers. In F. J. Brandenburg, editor, *Graph Drawing (Proc. GD '95)*, volume 1027 of *Lecture Notes in Computer Science*, pages 254–266. Springer-Verlag, 1996.

FK97. U. Fößmeier and M. Kaufmann. Algorithms and area bounds for nonplanar orthogonal drawings. In G. Di Battista, editor, *Graph Drawing (Proc. GD '97)*, volume 1353 of *Lecture Notes in Computer Science*, pages 134–145. Springer-Verlag, 1997.

IL97. C. Iturriaga and A. Lubiw. Elastic labels: The two–axis case. In G. Di Battista, editor, *Graph Drawing (Proc. GD '97)*, volume 1353 of *Lecture Notes in Computer Science*, pages 181–192, 1997.

KM98. G. W. Klau and P. Mutzel. Quasi–orthogonal drawing of planar graphs. Technical Report MPI–I–98–1–013, Max–Planck–Institut für Informatik, Saarbrüücken, May 1998.

KM99. G. W. Klau and P. Mutzel. Optimal compaction of orthogonal grid drawings. In G. P. Cornuéjols, editor, *Integer Programming and Combinatorial Optimization (IPCO '99)*, number 1610 in Springer Lecture Notes in Computer Science, pages 304–319, 1999.

KT97. K. G. Kakoulis and I. G. Tollis. An algorithm for labeling edges of hierarchical drawings. In G. Di Battista, editor, *Graph Drawing (Proc. GD '97)*, volume 1353 of *Lecture Notes in Computer Science*, pages 169–180, 1997.

Len90. T. Lengauer. *Combinatorial Algorithms for Integrated Circuit Layout.* John Wiley & Sons, New York, 1990.

Pat99. M. Patrignani. On the complexity of orthogonal compaction. Technical Report RT–DIA–39–99, Dipartimento di Informatica e Automazione, Università degli Studi di Roma Tre, January 1999.

Tam87. R. Tamassia. On embedding a graph in the grid with the minimum number of bends. *SIAM J. Comput.*, 16(3):421–444, 1987.

WS. A. Wolff and T. Strijk. The map labeling bibliography. `http://www.inf.fu-berlin.de/map-labeling/bibliography`.

Almost Bend-Optimal Planar Orthogonal Drawings of Biconnected Degree-3 Planar Graphs in Quadratic Time*

Ashim Garg[1] and Giuseppe Liotta[2]

[1] Department of Computer Science and Engineering
State University of New York at Buffalo, Buffalo, New York 14221, USA
agarg@cse.buffalo.edu

[2] Dipartimento di Ingegneria Elettronica e dell'Informazione
Universita' Di Perugia, Perugia, Italy
liotta@diei.unipg.it

Abstract. Let G be a degree-3 planar biconnected graph with n vertices. Let $Opt(G)$ be the minimum number of bends in any orthogonal planar drawing of G. We show that G admits a planar orthogonal drawing D with at most $Opt(G)+3$ bends that can constructed in $O(n^2)$ time. The fastest known algorithm for constructing a bend-minimum drawing of G has time-complexity $O(n^5 \log n)$ and therefore, we present a significantly faster algorithm that constructs almost bend-optimal drawings.

1 Introduction

An *orthogonal* drawing of a graph maps its vertices to points in the plane and its edges to a sequence of alternating horizontal and vertical line segments. A *planar* drawing is one with no edge-crossings. Orthogonal drawings have applications in a variety of fields such as Databases, Software Engineering, and VLSI design.

Bend-minimization is an important aesthetic criteria for orthogonal drawings. Several heuristics for bend-minimization are available. (see for example [7,4]). Garg and Tamassia [2] have shown that the bend-minimization for general planar graphs is NP-hard. Tamassia [6] has given an $O(n^2 \log n)$ time bend-minimization algorithm for embedded planar graphs. Later, Garg and Tamassia [3] improved the time-complexity of the algorithm to $O(n^{1.75} \log n)$. Di Battista, Liotta, and Vargiu [1] have given an $O(n^3)$ algorithm for constructing bend-minimum planar orthogonal drawings of series-parallel graphs.

In this paper, we study the problem of constructing bend-minimum planar orthogonal drawings of degree-3 planar graphs. Previously, Rahman, Nakano, and Nishizeki [5] have given a linear time algorithm for constructing a bend-minimum drawing of a triconnected cubic (each vertex has degree exactly 3)

* Research supported in part by a Reifler award, by the CNR Project "Geometria Computazionale Robusta con Applicazioni alla Grafica ed al CAD' and by the ESPRIT LTR Project 20244 (ALCOM-IT)

J. Kratochvíl (Ed.): GD'99, LNCS 1731, pp. 38–48, 1999.

plane graph. Di Battista et. al. [1] have given an $O(n^5 \log n)$ time algorithm for constructing bend minimum planar drawing of a degree-3 planar graph.

Our main result is as follows: Let G be a degree-3 planar biconnected graph with n vertices. Let $Opt(G)$ be the minimum number of bends in any orthogonal planar drawing of G. We show that G admits a planar orthogonal drawing D with at most $Opt(G)+3$ bends that can constructed in $O(n^2)$ time. The fastest known algorithm (of [1]) for constructing a bend-minimum drawing of G has time-complexity $O(n^5 \log n)$ and therefore, we present a significantly faster algorithm that constructs almost bend-optimal drawings.

Let G be a planar connected graph. The *degree* of a vertex of G is equal to the number of edges incident on it. The *degree* of G is equal to the maximum degree of a vertex of G. A *split pair* of G is either a pair of adjacent vertices of a a pair of vertices whose removal divides G into two or more connected graphs.

2 Spirality and Polar Drawings

Let G be a degree-3 planar connected graph. A *drawing* D of G is a mapping of its vertices to points in the plane and its edges to a set of alternating horizontal and vertical line-segments connecting its end vertices such that no two edges intersect each other. A *bend* B of an edge e of G in D is the common end point of two consecutive line-segments of e such that the angle between the line segments is not equal to $180°$.

Let D be a drawing of G. Let f be a face of D. Let l_1 and l_2 be two line-segments that appear consecutively in a clockwise traversal of the line-segments of f. Let v be a vertex or bend of f such that l_1 and l_2 are incident on v. Let θ be the counterclockwise angle between l_1 and l_2. We say that v *makes* an angle θ in f. Let $deg(f, \theta)$ denote the total number of vertices and bends that make an angle θ in f (those vertices that make angle θ more than once, for different pairs of consecutive line segments incident on them, will also get counted more than once). The following lemma can be derived easily from the results of [6]:

Lemma 1. *[6] Let D be a drawing of a planar connected graph G. Let f be an internal face of D. Then, $deg(f, 270°, D) - deg(f, 90°, D) + 2deg(f, 360°) = -4$.*

Let G be a connected planar graph with two distinguished vertices u, and v with degree at most 2, called its *poles*. A *polar drawing* of G is one in which both u and v are on its external face (see Figure 1). Let D be a polar drawing of G with external face f. f consists of two subpaths p_1 and p_2 connecting u and v, which are called its *contour paths*. Let $deg(p_1, \theta)$ denotes the total number of vertices and bends of p_1, except u and v, that make an angle θ in f. The *spirality* of p_1 in D is equal to $deg(p_1, 270°) - deg(p_1, 90°)$. [1] We likewise define the *spirality* of p_2 in D. The *spirality* of D is defined as equal to the minimum of the spiralities of p_1 and p_2.

[1] our definition of spirality is slightly different from that of [1].

D is a *diagonal* drawing of G if either G consists only of one vertex (in which case $u = v$), or each pole of G with degree 2 makes a 270° angle in f, and the spirality of both the contour paths of f is 1. (see Figure 1(a)).

D is a *side-on* drawing of G if G consists of at least two vertices, each pole of G with degree 2 makes a 270° angle in f, the spirality of one contour path of f is 0 and the spirality of the other contour path of f is either 0, 1, or 2 (see Figure 1(b)). A *bend-minimum* side-on drawing of G is one with the fewest bends amongst all the side-on drawings of G. We define the *bend-minimum* diagonal and *bend-minimum* polar drawings likewise. Notice that a side-on drawing has spirality 0 and a diagonal drawing has spirality 1.

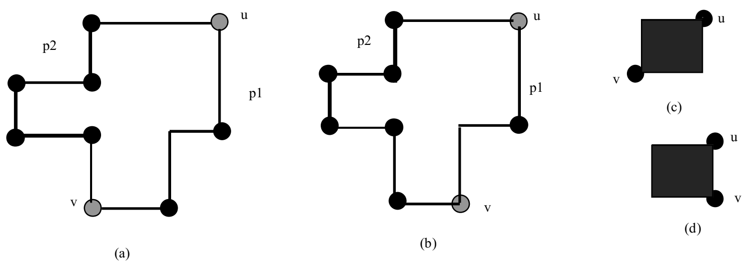


Fig. 1. Polar Drawings of a graph G with poles u and v: (a) A diagonal drawing D_1 of G; (b) A side-on drawing D_2 of G. p_1 and p_2 are the contour paths of the external face of D_1 and D_2. (c) A schematic representation of a diagonal drawing with poles u and v. (d) A schematic representation of a side-on drawing with poles u and v

Lemma 2. *Let G be a degree-3 connected planar graph with poles u and v. Then,*

- *G admits either a diagonal or a side-on drawing that has the minimum number of bends among all the polar drawings of G, and*
- *if G does not admit a side-on drawing that has the minimum number of bends among all the polar drawings of G, then G also does not admit any polar drawing with spirality less than 0 that has the minimum number of bends among all the polar drawings of G.*

3 Drawing Triconnected Cubic Plane Graphs with Minimum Bends

Rahman, Nakano, and Nishizeki [5] have given a linear time algorithm for constructing a bend-minimum drawing of a triconnected cubic plane graph. As we will see later, we use this algorithm for drawing an R node.

A *cubic graph* is one where each vertex has degree exactly 3. A *plane graph* is one with a fixed embedding and a fixed external face. A *drawing* of a plane graph is one that preserves the embedding and external face of the graph. Let G be a cubic triconnected plane graph. Let $C_0(G)$ denote the external face of G. Let G' be a plane graph obtained from G by inserting four dummy vertices,

called its *corner vertices*, in $C_0(G)$. We define a *k-legged cycle*, *descendent cycle*, *child cycle*, *corner cycle*, and *genealogical tree* of G, and G' as in [5]. Similarly, we define a *leg-vertex*, the *contour paths*, and *red* and *green* paths of a 3-legged cycles as in [5]. $C_0(G')$ consists of four *contour paths*, each connecting two corner vertices that are adjacent in a counterclockwise traversal of $C_0(G')$. Let C be a 3-legged cycle of G. C consists of 3 contour paths that connect two leg-vertices that are adjacent in a counterclockwise traversal of C. Let $G(C)$ denote the plane subgraph of G inside (and including) C. (See [5] for details). A *feasible drawing* D of G' is one such that D has the minimum number of bends among all the drawings of G', each corner-vertex of G' makes 270° angle in $C_0(G')$, and the spirality of each contour path of $C_0(G')$ is 0.

Let C be a 3-legged descendent cycle of G'. Let p be a contour path of C. A *feasible drawing* D of $G(C)$ *with respect to* p is one such that D has the minimum number of bends among all the drawings of $G(C)$, each leg-vertex of C makes 270° angle in the external face of D, the spirality of the two contour paths of C other than p is 0, and the spirality of p is 1.

Notice that [5] gives stronger definitions of feasible drawings, but for our purposes, the above definitions are sufficient. A *rectangular drawing* D is one in which each edge is drawn as a single line-segment and each face is drawn as a rectangle. D has exactly four vertices, called its *corners*, that make 270° angles in its external face.

We describe below a small variation of the algorithm of [5], which we call `Algorithm LinearDraw(G)`, that constructs a bend-minimum drawing D of a cubic triconnected plane graph G in linear time:

`Algorithm LinearDraw(`G`):`

1. Find as many as and up to 4 independent corner cycles $L_1, L_2, \dots, L_k$ (where $k \le 4$) of G. For each L_i, insert one dummy vertex l_i in an edge common to $C_0(G)$ and a green path of C_i. Let G^* be the plane graph thus obtained. If k is less than 4, then insert $4-k$ more dummy vertices $l'_1, '_2, \dots, l'_{4-k}$ in to the edges of $C_0(G^*)$ such that overall, at most two dummy vertices get inserted in to the same edge of $C_0(G)$. Let G' be the plane graph thus obtained. G' has four corner vertices $l_1, \dots, l_k, l'_1, \dots, l'_{4-k}$.
2. Let $C_1, C_2, \dots, C_m$ be the child cycles of G'. Collapse each C_i into a super node $S(C_i)$. Let Let G'' be the plane graph with no 3-legged cycles thus obtained.
3. Construct a rectangular drawing $D(G'')$ of G'' with the corner vertices of G'' (i.e., vertices $l_1, \dots, l_k, l'_1, \dots, l'_{4-k}$) as its corners.
4. For each C_i, invoke `Algorithm FeasibleDraw(`C_i, p`)`, where p is a green path of C_i, to construct a feasible drawing $D(C_i)$ with respect to p of $G(C_i)$.
5. Patch each $D(C_i)$ into drawing $D(G'')$ at $S(C_i)$ without introducing any additional bends to get a drawing $D(G')$ of G'.
6. In $D(G')$, replace vertices $l_1, \dots, l_k, l'_1, \dots, l'_{4-k}$ by bends to obtain a bend-minimum drawing D of G.

Given a 3-legged cycle C with leg-vertices a, b, and c, and a green path p of C, `Algorithm FeasibleDraw(`C, p`)` constructs a feasible drawing of C with

respect to p. It is implemented similar to `Algorithm LinearDraw`, except that in the Step 1, we insert exactly one dummy vertex d, and designate a, b, c and d as the corner vertices of the graph $G'(C)$ thus obtained. The vertex d is inserted in an edge of p.

Lemma 3. *Let H be a cubic triconnected plane graph. Let $e = (u, v)$ be a distinguished edge of G on its external face, called its* reference edge. *Let G be a plane graph with* poles u and v, *and external face $C_0(G)$, obtained from H by removing e and inserting one or more degree-2 vertices in some edges of H. Suppose n is the number of vertices in G. Then, we can construct in $O(n)$ time, a bend-minimum diagonal or side-on drawing D of G, where D is a diagonal drawing if and only if G does not admit a side-on drawing with less than or equal number of bends than D.*

Sketch of Proof:. We construct D using `Algorithm LinearDraw`(G) after making two modifications in it:

1. If a 3-legged cycle C contains a degree-2 vertex z, then z can be designated a corner vertex of C. This obviates the need to insert a new dummy vertex in an edge of C in Step 1 of `Algorithm FeasibleDraw`. We, therefore, change the definition of a *green path* of a 3-legged cycle C as follows:
   - If C does not contain any child cyle, or no child cycle of C has a green path that has an edge in common with C, then
     - if none of the three contour paths contain a degree-2 vertex, then all the three contour paths are categorized as green paths, otherwise
     - a contour path of C is categorized as a green path if and only it contains a degree-2 vertex,
   - if C contains at least one child cycle that has a green path that has an edge in common with C, then a contour path of C is categorized as a green path if and only if either it has an edge in common with a green path of a child cycle, or it contains a degree-2 vertex.

   Correspondingly, in Step 1 of `Algorithm FeasibleDraw` (C, p) if p contains a degree-2 vertex z, then we designate z as a corner vertex instead of inserting a new dummy vertex d in an edge of p. Since d, if inserted, would have appeared as a bend in the final drawing, designating z as a corner vertex instead of inserting d saves us one bend. Using a similar proof of [5], it can be shown that `Algorithm FeasibleDraw` (C, p) will construct a feasible drawing of C with respect to p in linear time.
2. We change Step 1 of `Algorithm LinearDraw` as follows: We can designate u and v as two corner vertices of G'. We still need to designate two more corner vertices for G'. Let us denote these corner vertices by a_1 and a_2. Let p_1 and p_2 be the two subpaths of $C_0(G)$ with end points u and v. We say that the corner vertex a_j, where $1 \leq j \leq 2$, gets *assigned* to path p_i, , where $1 \leq i \leq 2$, if we designate as a_j, either a degree-2 vertex of p_i, or a dummy vertex that we insert into an edge of p_i during the execution of Step 1. In a side-on drawing of G with poles u and v, both a_1 and a_2 get assigned to the same path, either p_1 or p_2, and in a diagonal drawing, one each is assigned

to p_1 and p_2. To determine the path to which we should assign each a_j, for each p_i, we count how many degree-2 vertices (excluding u and v) are there in p and how many corner cycles have a green path that has at least one edge in common with p_i (here we extend the definition of a corner cycle to include 2-legged cycles of C. Notice that, since H is triconnected, it can be shown that such cycles contain either u or v.) . Based on the counts, we assign each a_j to a p_i such that we get either a bend-minimum side-on or a bend-minimum diagonal drawing D of G, where D is a diagonal drawing if and only if G does not admit a side-on drawing with less than or equal number of bends than D. Details will be given in the full paper.

4 SPQR Tree

Let G be a planar biconnected graph. Let $e = (s, t)$ be an edge of G. An *SPQR* decomposition of G with *reference* edge e is its recursive decomposition into components of four types: S, P, Q, and R, where the initial decomposition divides G at the split pair $\{s, t\}$. Each *SPQR* decomposition corresponds to an *SPQR* decomposition tree τ. τ consists of four types of nodes- S, P, Q, and R, which correspond to the S, P, Q and R components, respectively, of the decomposition. We can always always orient the edges of τ such that the root of τ is a Q node that corresponds to the edge e. Each node of τ corresponds to a subgraph of G, called its *pertinent graph*. In particular, the pertinent graph of a Q node consists of a single edge of G. The pertinent graph of each node X of τ has two distinguished vertices called its *poles*. Notice that the poles of both the root of τ and its child are s and t. Also associated with each node X of τ is a graph, called its *skeleton*, and denoted as *skel(X)*. The skeleton of an R node is a triconnected graph, of a P node is a bundle of parallel edges, of an S node is a chain of edges, and of a Q node is a single edge. See [1] for details.

Let X be an S node of τ. Let u and v be the parent split vertices of X. We can order the children of X as $C_1, C_2, \ldots, C_{k-1}, C_k$, where the child component C_1 is incident on u, child component C_2 shares a vertex with C_1, C_3 shares a vertex with C_4, and so on, and finally, C_k shares a vertex with C_{k-1} and also, C_k is incident on v. We call C_1 and C_k the *extreme children* of X, and the ordering $C_1, C_2, \ldots, C_{k-1}, C_k$ a *canonical ordering* of the children of X.

Let G be a biconnected degree 3 planar graph. Let τ be an *SPQR* decomposition tree of G with reference edge e. Let X be a node of τ. The following facts can be easily derived from the fact that each vertex of G has degree at most 3:

Fact 1 If X be an S node of τ. Let $C_1, C_2, \ldots, C_{k-1}, C_k$ is a canonical ordering of the children of X. We have that, if C_i is a non-Q node, then C_{i-1} and C_{i+1} are Q nodes. In other words, for each non-Q child of X, the children of X before and after it in the canonical ordering are Q nodes, i.e., correspond to single edges of G. More over, if the parent of X is not the root of τ, then both C_1 and C_k are Q nodes, i.e., correspond to single edges of G. Also, we can always construct a τ such that each C_i is either a P node, Q node, or an R node.

Fact 2 Each P-node of τ has exactly two children which are either S or Q nodes.

Fact 3 Each child of an R-node of τ is either an S node or a Q node.

We define the core graph and pole of a node X of τ as follows: If X is a P node, Q node, R node, or an S node whose parent is the root of τ, then its *core graph* is the same as its pertinent graph, and the *pole* of its core graph is the same as the pole of its pertinent graph.

If X is an S node whose parent is not the root of τ, then let $C_1, C_2, \ldots, C_{k-1}, C_k$ be a canonical ordering of the children of X. Let u and v be the poles of the pertinent graph of X. From Fact 1, C_1 and C_k are Q-nodes, i.e., they correspond to single edges of G. Suppose C_1 and C_k correspond to edges (u, a) and (b, v), respectively, of G. Let $H(C_i)$ denote the core graph of C_i. Since each C_i is a P, Q, or R node, $H(C_i)$ is the same as the pertinent graph of C_i. The *core graph* H of X is defined as the subgraph of the pertinent graph of H that consists of the graphs $H(C_2), H(C_3), \ldots, H(C_{k-1})$, i.e., $H = H(C_2) \bigcup H(C_3) \bigcup H(C_{k-1})$. In other words, the core graph of X is the graph obtained by removing edges (u, a) and (b, v) from the pertinent graph of X. Vertices a and b are designated the *poles* of H.

The *poles* of X are the same as the poles of its core graph.

Lemma 4. *Let G be a degree-3 planar graph. Let τ be an* SPQR *tree of G that corresponds to an* SPQR *decomposition of G with a reference edge e. Let X be a non-root node of τ, whose pertinent graph has n vertices. Then, we can construct in $O(n)$ time, a bend-minimum polar drawing D of the core graph H of X such that:*

1. *if X is an S node whose core graph consists of at least two vertices,or is a Q node, then D is a side-on drawing,*
2. *if X is an S node whose core graph consists of exactly one vertex, then D is a diagonal drawing,*
3. *if X is a P, or an R node, then D is either a diagonal or a side-on drawing, and*
4. *D is a diagonal drawing if and only if H does not admit a side-on drawing with less than or equal number of bends than D.*

Sketch of Proof:. Our proof is constructive. Starting with the leaves of τ, for each node X of τ, we construct a side-on or diagonal drawing of the core graph of X with properties as given in the statement of the lemma.

Let X be a non-root node of τ. Let H be the core graph of X. Let u and v be the poles of X. Let $D(X)$ denotes the drawing—side-on or diagonal—of H constructed by our proof. We consider the following cases:

- X is a Q node: Then, H is a single edge $t = (u, v)$. We draw t as a single line-segment.
- X is an S node: We have two subcases:
  - H consists of a single vertex $a = u = v$: Then, we draw a as a point, which by the definition of a diagonal drawing, is a diagonal drawing.

- H consists of more than one vertex: Let $C_1, C_2, \ldots, C_{k-1}, C_k$ be a canonical ordering of the children of X. Recall that if the parent of S is not the root of τ, then $H = H(C_2) \bigcup H(C_3) \bigcup H(C_{k-1})$, where $H(C_i)$ is the core graph of C_i. We have three cases, depending on whether both $D(C_2)$ and $D(C_{k-1})$ are side-on drawings, or are both diagonal drawings, or one is a side-on and the other is a diagonal drawing. As shown in Figure 2, in all three cases, by simply stacking the drawings $D(C_2), D(C_3), \ldots, D(C_{k-1})$ one above the other in that order, we can construct a side-on drawing $D(X)$. Since each $D(C_i)$ is a bend-minimum polar drawing of $H(C_i)$, and we do not insert any new bends while stacking them, it follows that $D(X)$ is a bend-minimum polar drawing of H. If the parent of S is the root of τ, then we can construct $D(X)$ in a similar fashion by vertically stacking drawings $D(C_1), D(C_2), \ldots, D(C_k)$.

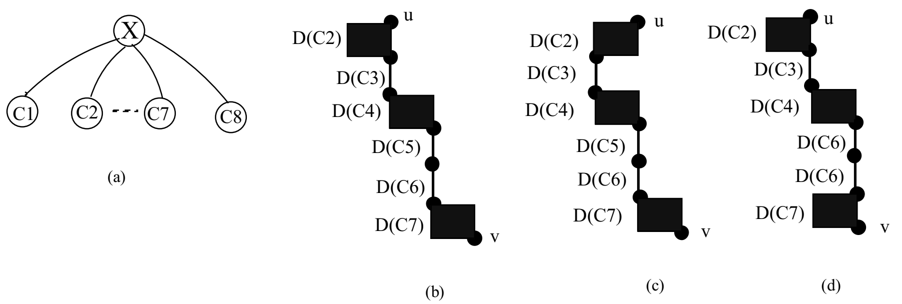


Fig. 2. Constructing $D(X)$ when X is an S node such that the parent of X is not the root node: (a) An S node X with 8 children $C_1, C_2, \ldots, C_8$ of which only C_2, C_4 and C_7 are non-Q nodes; (b,c,d) Constructing $D(X)$ from $D(C_2), D(C_3), \ldots, D(C_7)$: (b) When $D(C_2)$ is a side-on and $D(C_7)$ is a diagonal drawing; (c) When both $D(C_2)$ and $D(C_7)$ are diagonal drawings; (d) When both $D(C_2)$ and $D(C_7)$ are side-on drawings. Vertices u and v are the poles of X.

– X is a P node: From Fact 2, X has two children C_1 and C_2, and they are either S or Q nodes. We have two subcases:
  - C_2 is a Q node: Let $H(C_1)$ be the core graph of C_1. Suppose C_2 corresponds to a single edge $t = (u, v)$ of G. If $D(C_1)$ is a side-on drawing, then we can construct $D(X)$ from $D(C_1)$ and $D(C_2)$ as shown in Figure 3(a) without adding any new bends. Since, $D(C_1)$ is a bend-minimum polar drawing of $H(C_1)$, it follows that $D(X)$ is also bend-minimum polar drawing of the core graph of X. If $D(C_2)$ is a diagonal-drawing, then we can construct $D(X)$ from $D(C_1)$ and $D(C_2)$ as shown in Figure 3(b) by adding one more bend. To show that $D(X)$ is also a bend-minimum polar drawing of H, consider a bend-minimum polar drawing D of H (also see Figure 3(c)). D contains a polar subdrawing D' of the core graph of C_1. Since $D(C_1)$ is a bend-minimum polar drawing, D' has at

least as many bends as $D(C_1)$. Hence, if any of the edges (a, u), (b, v) and t have a bend in D, then we are done, otherwise, consider the face f of D that contains the edges (a, u), (b, v), and t, where a and b are the poles of $H(C_1)$ (see Figure 3(c)). f also contains a contour path p_1 of the external face of D'. From Lemma 1, it follows that if face f does not have any bend in edges (a, u), (b, v), or t, then p_1 must have spirality at most 0, and hence D' must have spirality at most 0. However, then, since Lemma 4(4) holds for $D(C_1)$, it follows from Lemma 2 that D' has at least one bend more than $D(C_1)$. That is, D has at least as many bends as $D(X)$, and therefore $D(X)$ is also a bend-minimum polar drawing.

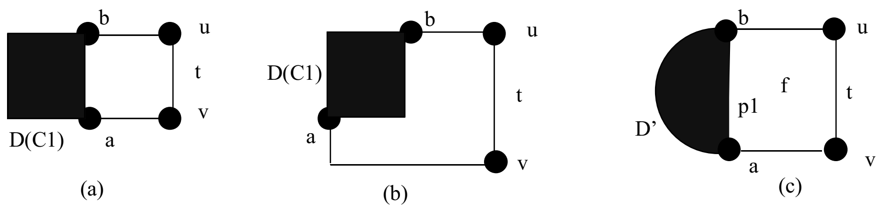


Fig. 3. Constructing $D(X)$ for a P node X with children C_1 and C_2, where C_2 corresponds to a single edge $t = (u, v)$: (a) When $D(C_1)$ is a side-on drawing, (b) When $D(C_1)$ is a diagonal drawing. In both cases, $D(X)$ is a side-on drawing; (c) Proof of the bend-optimality of $D(X)$. a and b are the poles of C_1, and u and v are poles of X.

  - C_2 is not a Q node: If both $D(C_1)$ and $D(C_2)$ are side-on drawings, than we construct a bend-minimum side-on drawing $D(X)$ as shown in Figure 4(a). If at least one of $D(C_1)$ and $D(C_2)$ is a diagonal drawing, then we can construct a diagonal drawing $D(X)$ as shown in Figure 4(b,c). In both the cases, we do not add any bends, and hence, $D(X)$ is bend-minimum side-on or diagonal drawing. Using a reasoning similar to one for the previous case, where C_2 is a Q node, we can show that when $D(X)$ is a diagonal drawing, then X does not admit any side-on drawing with less than or equal number of bends than $D(X)$.

– X is a R node: Let r be the reference edge of $skel(X)$. From Fact 3, each child of X is either an S node or a Q node. We first remove r from $skel(X)$. Next, for each edge k of $skel(X)$ that corresponds to an S child of X whose core graph consists of at least two vertices, we insert two dummy vertices in k. Also, for each edge k of $skel(X)$ that corresponds to an S child of X whose core graph consists of exactly one vertex, we insert one dummy vertex in k. Let L be the graph thus obtained. We designate u and v as the poles of L. We construct a bend-minimum side-on or diagonal drawing D of L using Lemma 3. From D, we construct $D(X)$ without adding any new bends as follows: Let k be an edge of $skel(X)$ that corresponds to an S child C of X. let $H(C)$ be the core graph of C. We have two cases:

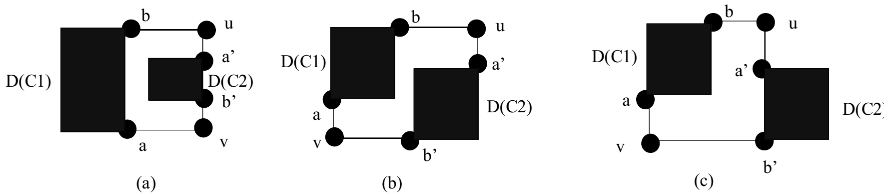


Fig. 4. Constructing $D(X)$ for a P node with children C_1 and C_2, neither of which is a Q node: (a) When both $D(C_1)$ and $D(C_2)$ are side-on drawings; (b) When both $D(C_1)$ and $D(C_2)$ are diagonal drawings; (c) When $D(C_1)$ is a diagonal drawing and $D(C_2)$ is a side-on drawing. a and b are the poles of C_1, a' and b' are the poles of C_2, and u and v are the poles of X.

- $H(C)$ consists of a single vertex z: Let a be the dummy vertex introduced in k to obtain L from $skel(X)$. We simply replace a by z.
- $H(C)$ consists of at least two vertices: Let a and b be the dummy vertices introduced in k to obtain L from $skel(X)$. Since $D(C)$ is a side-on drawing (because C is an S node) irrespective of the angles made by a and b in their incident faces in D, we can replace a and b and the edge (a, b) by $D(C)$ without adding any new bends.

Lemma 5. *Let G be a biconnected degree-3 planar graph with n vertices. Let $e = (u, v)$ be an edge of G. Let M be the minimum number of bends in any drawing of G with e on the external face. Then, we can construct in $O(n)$ time, a drawing D of G with e on the external face such that D has at most $M + 3$ bends.*

Sketch of Proof:. Let τ be an *SPQR* decomposition tree of G with reference edge e. As mentioned earlier in Section 4, the root X of τ is a Q node that corresponds to e. Let C be the child of X with core graph $H(C)$. u and v are the poles of $H(C)$. Let $D(C)$ be the bend-minimum polar drawing of $H(C)$ constructed using Lemma 4. As shown in Figure 5, from $D(C)$, we can construct a drawing D of G with e on external face by adding at most 3 more bends irrespective of whether $D(C)$ is side-on or diagonal. Since a bend-minimum drawing of G with e on external face contains a polar drawing of $H(C)$ as a subdrawing, and $D(C)$ is bend-minimum polar drawing of $H(C)$, it follows that D has at most $M + 3$ bends. Since, from Lemma 4, $D(C)$ can be constructed in $O(n)$ time, it follows that D can also be constructed in $O(n)$ time.

5 Main Theorem

Theorem 1. *Let G be a biconnected degree-3 planar graph with n vertices. Let $Opt(G)$ be the minimum number of bends in any drawing of G. Then, we can construct in $O(n^2)$ time, a drawing D of G such that D has at most $Opt(G) + 3$ bends.*

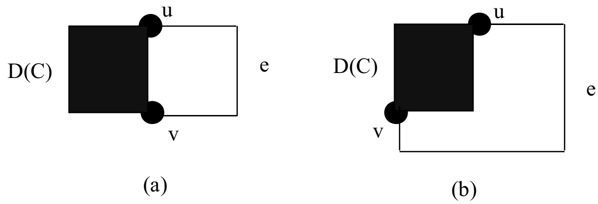


Fig. 5. Proof of Lemma 5: Constructing a drawing for the root node X from $D(C)$, where C is the child of X: (a) when C is a side-on drawing, and (b) when C is a diagonal drawing. e is the reference edge.

Sketch of Proof:. For each edge e of G, we construct a drawing using Lemma 5 that has e on the external face. The drawing with minimum number of bends among these drawings will have at most $Opt(G) + 3$ bends. Since G has $O(n)$ edges, and constructing each drawing takes $0(n)$, the total running time is $O(n^2)$.

References

1. G. Di Battista, G. Liotta, and F. Vargiu. Spirality of orthogonal representations and optimal drawings of series-parallel graphs and 3-planar graphs. In *Proc. Workshop Algorithms Data Struct.*, volume 709 of *Lecture Notes Comput. Sci.*, pages 151–162. Springer-Verlag, 1993.
2. A. Garg and R. Tamassia. On the computational complexity of upward and rectilinear planarity testing. In R. Tamassia and I. G. Tollis, editors, *Graph Drawing (Proc. GD '94)*, volume 894 of *Lecture Notes Comput. Sci.*, pages 286–297. Springer-Verlag, 1995.
3. A. Garg and R. Tamassia. A new minimum cost flow algorithm with applications to graph drawing. In S. C. North, editor, *Graph Drawing (Proc. GD '96)*, Lecture Notes Comput. Sci. Springer-Verlag, 1997.
4. A. Papakostas and I. G. Tollis. Improved algorithms and bounds for orthogonal drawings. In R. Tamassia and I. G. Tollis, editors, *Graph Drawing (Proc. GD '94)*, volume 894 of *Lecture Notes Comput. Sci.*, pages 40–51. Springer-Verlag, 1995.
5. M. S. Rahman, S. Nakano, and T. Nishizeki. A linear algorithm for optimal orthogonal drawings of triconnected cubic planar graphs. In G. D. Battista, editor, *Graph Drawing (Proc. GD '97)*, volume 1353 of *Lecture Notes in Computer Science*, pages 99–110. Springer Verlag, 1998.
6. R. Tamassia. On embedding a graph in the grid with the minimum number of bends. *SIAM J. Comput.*, 16(3):421–444, 1987.
7. R. Tamassia and I. G. Tollis. Planar grid embedding in linear time. *IEEE Trans. Circuits Syst.*, CAS-36(9):1230–1234, 1989.

Fully Dynamic 3-Dimensional Orthogonal Graph Drawing*

M. Closson, S. Gartshore, J. Johansen, and S. K. Wismath[1]

Department of Mathematics and Computer Science,
University of Lethbridge,
Lethbridge, Alberta, T1K-3M4, Canada.
wismath@cs.uleth.ca

Abstract. In a 3-dimensional orthogonal drawing of a graph, vertices are mapped to grid points on an integer lattice and edges are routed along integer grid lines. In this paper, we present a layout scheme that draws any graph with n vertices of maximum degree 6, using at most 6 bends per edge and in a volume of $O(n^2)$. The advantage of our strategy over other drawing methods is that our method is fully dynamic, allowing both insertion and deletion of vertices and edges, while maintaining the volume and bend bounds. The drawing can be obtained in $O(n)$ time and insertions/deletions can be performed in $O(1)$ time. Multiple edges and self loops are permitted. A more elaborate construction that uses only 5 bends per edge, and a simpler, more balanced layout that requires at most 7 bends per edge are also described.

1 Introduction and Previous Work

In this paper we describe a 3-dimensional orthogonal layout strategy for graphs of maximum degree 6. In the final layout, vertices and edges lie on a 3-dimensional integer grid; in particular, vertices occur at grid points of the form (x', y', z'), where $x', y', z' \in Z$, and edges are routed along integer-valued grid lines, possibly bending at integer grid points. Each edge (u, v) joins the points representing vertices u and v and intersects no other vertex point. No pair of edges intersect (except at endpoints).

There have been several 3-dimensional layout strategies proposed in the graph drawing literature for graphs of maximum degree 6 - see [1], [3], [4], [5], [6]. Drawings of graphs of higher degree have been investigated in [2] and [5]. For the most part, these results have focussed on the trade-offs among: overall volume of the resulting layout, the maximum number of bends on an edge, the maximum length of an edge, *etc.* Experimental comparisons can be found in the paper by Di Battista, Patrignani, and Vargiu [3].

Although there are some classes of graphs of maximum degree 6 that can be drawn with few bends (e.g. trees can be drawn with 0 bends), the introduction of multiple edges, or self loops immediately implies that three bends per edge

* N.S.E.R.C. is gratefully thanked for financial assistance.

J. Kratochvíl (Ed.): GD'99, LNCS 1731, pp. 49–58, 1999.

may be required. For a self loop, the necessity of three bends is evident. A simple but tedious case study can be used to demonstrate that a pair of vertices with 6 edges between them, can not be drawn with at most two bends per edge.

Our result is motivated primarily by a dynamic version of the layout problem in which edges and vertices of the graph can be inserted or deleted over time. Papakostas and Tollis [5] introduced a semi-dynamic solution in which only vertices are permitted to be inserted and at all stages, the resulting layout must represent a connected graph. Our layout strategy is *fully dynamic*, allowing insertion and deletion of vertices and edges. The volume of the layout is bounded by $O(n^2)$, each edge has at most 6 bends, and insertions and deletions are accomplished in $O(1)$ time. The dynamic nature of our layout relies on the fact that at any time, a free *port* on any vertex may safely be connected to a free port of any other vertex without disturbing the layout, and the connection requires at most 6 bends.

Multiple edges and self-loops are also accommodated in our drawings. Multiple edges may prove to be an important consideration in applications that require some degree of "fault-tolerance". Previous 3-dimensional orthogonal constructions have typically been restricted to simple graphs and can not support dynamic operations. For example, one of the classic drawing algorithms in this area is due to Eades, Symvonis and Whitesides [4]. Their algorithm achieves $O(n^{1.5})$ volume but uses 7 bends, requires a matching and colouring pre-processing step, and is only suitable in a static context. The same general technique has recently been modified by the authors to examine volume/bend tradeoffs and in particular they achieve a layout with at most 6 bends per edge and $O(n^2)$ volume.

One final feature of our layout that may be useful in some applications, is that the width of the layout is constant - in fact the entire drawing lies on 7 planes between the planes $Y = -3$ and $Y = +3$.

2 Drawing Graphs in $O(n^2)$ Volume with 6 Bends

2.1 Introduction and Definitions

Initially, we describe our layout strategy in a static context, in which the entire graph (of maximum degree 6) is available. As will be seen, a dynamic version of the problem follows easily.

Let the vertices of the given graph be $v_1, v_2, v_3, ...v_n$ (labelled in an arbitrary order). The vertices will be located in a stair-case fashion, separated by an appropriate amount to permit the crossing-free embedding of the edges. More formally, vertex v_i is located at $(7i, 0, 5i)$. For convenience, we denote these coordinates as $(X(v_i), Y(v_i), Z(v_i))$. Each vertex may have at most 6 incident edges attached at **ports** which will be denoted as N, S, E, W, T, B - refer to figure 1. When viewed from above (*i.e.* from $+\infty$ along the Z axis), the compass designations effectively describe the ports.

Containing each vertex v is a small box of dimensions 7x6x5, called its **neighbourhood** and denoted by **N(v)**. Contained entirely inside N(v) are 6 **pedestals** - one for each port of v. Each pedestal consists of a line segment parallel

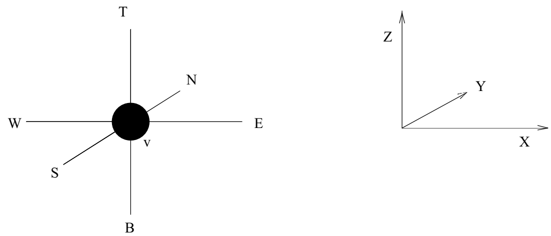


Fig. 1. The 6 ports on a vertex

to the Z axis. The **Bottom pedestal** is directly attached to v beneath it, and is of length 1. Similarly, the **Top pedestal** attaches to the top port of v and is of length 3. The **North pedestal** is of length 2 and extends from $(X(v), +1, Z(v))$ to $(X(v), +1, Z(v) + 2)$. The **South pedestal** is of length 1 and extends from $(X(v), -1, Z(v))$ to $(X(v), -1, Z(v) + 1)$. The **West** and **East pedestals** are degenerate points at $(X(v) - 1, 0, Z(v))$ and $(X(v) + 1, 0, Z(v))$ respectively; they are introduced for consistency in the description and will be modified in a subsequent section describing a related layout strategy.

The Neighbourhood N(v) consists of 5 layers in the Z-dimension, the:

- **Top layer** located on the plane $Z = Z(v) + 3$,
- **North layer** located on the plane $Z = Z(v) + 2$,
- **South layer** located on the plane $Z = Z(v) + 1$,
- **East, West, v layer** located on the plane $Z = Z(v)$,
- **Bottom layer** located on the plane $Z = Z(v) - 1$.

Edges are always considered as directed from the lower indexed vertex to the higher. Each pedestal is used only by *outgoing* edges from a vertex. *Incoming* edges are routed along **pillars**. The 6 pillars of a vertex v extend below the neighbourhood of v to the plane $Z = -1$. More precisely, the

- **North pillar** extends from $(X(v), 2, -1)$ to $(X(v), 2, Z(v) + 2)$,
- **South pillar** extends from $(X(v), -2, -1)$ to $(X(v), -2, Z(v) + 2)$,
- **East pillar** extends from $(X(v) + 2, 0, -1)$ to $(X(v) + 2, 0, Z(v) + 2)$,
- **West pillar** extends from $(X(v) - 2, 0, -1)$ to $(X(v) - 2, 0, Z(v) + 2)$,
- **Bottom pillar** extends from $(X(v), 0, -1)$ to $(X(v), 0, Z(v))$,
- **Top pillar** extends from $(X(v) - 3, 0, -1)$ to $(X(v) - 3, 0, Z(v) + 3)$.

Note that the four side pillars (N, S, E, W) extend to $Z(v)+2$ to accommodate self loops as will be described in a later section.

We refer to the orthogonal box directly below a vertex v (and containing the pillars of v), as the **airspace of** v. Edges pass safely through airspaces, by traversing *lanes*, which are guaranteed to avoid pillars. In particular, note that any line segment lying in the planes $Y = 1, Y = 3, Y = -1$, or $Y = -3$ intersects no pillar of *any* vertex.

Typically, an edge from a vertex v to a vertex w leaves v from the specified port, travels to the associated pedestal, climbs the pedestal to the appropriate

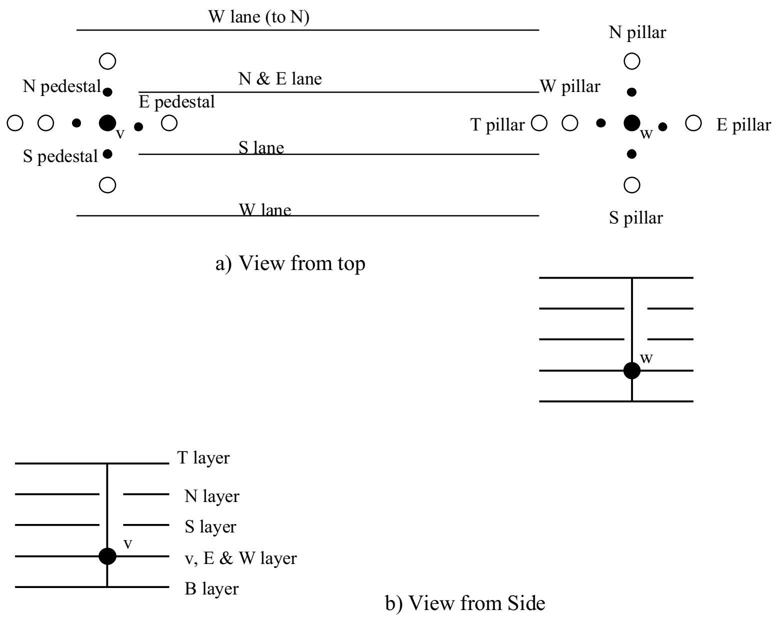


Fig. 2. Two Neighbourhoods

layer, is routed *on that layer* to the airspace under w along one of the four lanes, joins to the pillar associated with the destination port of w, climbs this pillar and finally enters w. Although the routings differ slightly for each port to port pair, this general technique applies, the critical observation being that the routing from the airspace of v to the airspace of w is done at some *layer of* v (*i.e.* the lower vertex). By using one of the four lanes, this section of the edge passes safely through the airspace of any intervening vertices without intersecting any of its pillars (which may have edges routed from below). Since the E and W ports of v share the same layer as v, special care is required to ensure routings from these ports that do not intersect each other or possible incoming edges to the N or S ports. Figure 3 shows the routing of two edges from v to w: a $W \to N$ connection and a $E \to E$ connection.

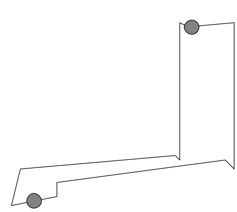


Fig. 3. W to N and E to E Routes

A connection to the Top port of vertex w comes up the Top pillar, along w's Top layer, and then directly down to w, thus safely passing above w's W pillar. For completeness, the 36 port-to-port routings are specified in Appendix A. Note that of these routings, 9 require 6 bends, 22 require 5 bends, and 5 require 4 bends.

2.2 Proof of Correctness

Lemma 1. *No pair of edges in the construction intersect.*

Proof:

Appendix A defines the paths for all port-to-port connections between a vertex v (at the lower level) to a vertex w ($\neq v$). To prove that edge (v, w) intersects no other edge in the layout, note that the routing of each edge consists of three portions:

1. inside v's neighbourhood
2. along a lane through intervening airspaces
3. in w's airspace.

Since the four lanes intersect no pillars, the second portion of edge (v, w) intersects no other edge.

Consider the portion of the edge (v, w) inside v's neighbourhood. Each of the 6 possible routings out of v intersect no pillar of v and so can not intersect any *incoming* edge to v. And similarly, the portion in w's airspace can intersect no outgoing edge from w.

It remains to consider the intersection of (v, w) with some outgoing edge from v (inside v's neighbourhood) or with some other incoming edge to w (inside w's airspace). Without loss of generality, we consider the case of (v, w) intersecting another edge between v and w but with different port assignments.

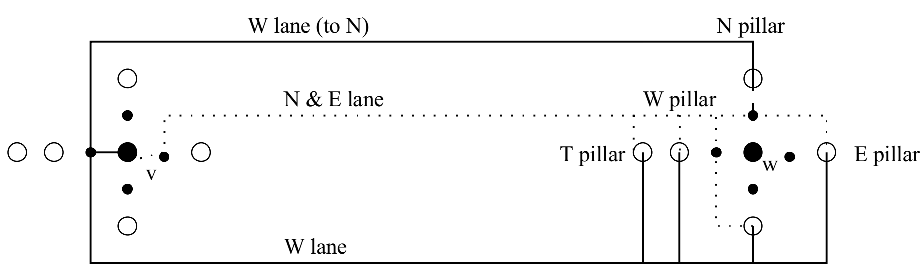


Fig. 4. E and W routings do not intersect

Since the N,S,T, and B ports each have their own planes inside the neighbourhood of v, it is simple to verify that no intersections are possible if either of the two edges involve these ports at v. However, the E and W ports share a plane

with v itself, and it is therefore critical to check that none of the 30 possible pairs involving these two ports from v intersect. In figure 4, the 6 routings from W are shown as solid lines and the 6 routings from E are displayed as dotted lines; note that only the routings to the pillars in w are shown. That no pair of solid and dotted paths intersect either in the neighbourhood of v nor in the airspace of w is easily determined. •

3 Dynamic Case

The layout strategy described in the previous section easily adapts to use in a dynamic context, in which both edges and vertices may be deleted or inserted. Insertion of a new vertex v_t is performed by placing it at $(7t, 0, 5t)$. It is clear that a new edge (u, v) can be added to the layout in $O(1)$ time - any free port on u can be connected to a free port on v, and furthermore, the overall volume of the layout remains the same. Deletion of a vertex is accomplished by replacing it by the top-most vertex and re-routing the (at most 6) incident edges. Let $v_1, v_2, ...v_t$ be the vertices in the layout at time t. To delete vertex v_i, the edges on v_i are first deleted. Denote by $w_1, w_2, ...w_6$ the vertices adjacent to vertex v_t. These edges are all deleted in the layout and vertex v_i is connected to $w_1, w_2, ...w_6$, using arbitrary ports on vertex v_i. Vertex v_t is then removed from the layout. The resulting layout has a volume of $6 \cdot 7 \cdot 5(t-1)^2$. Note that we measure the number of grid points rather than the length in each dimension.

Theorem 1. *Insertion or deletion of vertices or edges can be accomplished in* $O(1)$ *update time, and at all times, the volume of the layout is* $6 \cdot 7 \cdot 5 \cdot t^2$*, where* t *is the number of vertices in the layout.*

4 Variants

In this section, we consider two related constructions - a *spiral* layout that has a more balanced appearance, and a more elaborate version of the *stair-case* model that requires at most 5 bends per edge.

4.1 Achieving 5 Bends per Edge

In the stair-case layout there are only 9 port-to-port routings that require 6 bends per edge. The layout can be modified to achieve at most 5 bends per edge, however the routings become more involved. The volume bounds and dynamic nature of the algorithm are preserved.

4.2 Spiral Layout

One possible criticism of the strategy described in the previous sections is that the dimensions of the resulting layout are "unbalanced" - the construction is very narrow, with a Y dimension consisting of exactly 7 planes. Although there

may be practical applications of this feature of our strategy, from an aesthetic standpoint a more balanced layout may be preferable. In this section, we outline a strategy that produces a layout of dimensions $O(\sqrt{n}) \cdot O(\sqrt{n}) \cdot O(n)$, but with a slight penalty on the number of bends: 7 bends are required for some port-to-port connections. The dynamic nature of the previous strategy is preserved; an implementation and a short video describing the routing is available at `www.cs.uleth.ca/~wismath/threed`.

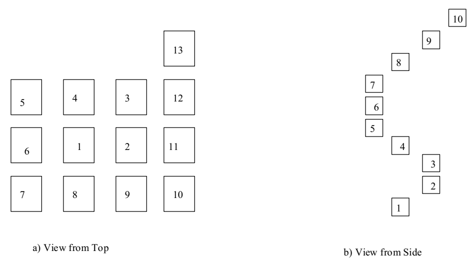


Fig. 5. Spiral layout: a) from above; b) from the side

Instead of placing the vertices in a linear staircase fashion, the vertices are embedded in an orthogonal spiral manner. When viewed from above, the vertices appear in a $\sqrt{n} \cdot \sqrt{n}$ grid as displayed in figure 5. However, as in the previous method, each vertex is assigned to a unique Z-plane, thus forming a spiral of linear height. Associated with each vertex are 7 planes (one for each port, and one for the vertex itself). The degenerate E and W pedestals described in section 2.1 are extended from the v layer to the appropriate plane. The general edge routing technique remains the same. An edge from a vertex v to vertex w, exits a port on v, climbs a pedestal to the appropriate plane, traverses a lane to correct in the X-coordinate, traverses a lane to correct in the Y-coordinate to enter w's airspace, climbs the appropriate pillar, and finally enters w at the specified port.

A more detailed account of this spiral layout scheme will be provided in the journal version of the paper.

4.3 Self-Loops

The general layout strategy for self loops is similar: from a port to a pedestal, up to the appropriate level, across to the specified pillar, down the pillar and into the vertex. A more detailed description and proof of correctness will be included in the journal version of the paper. A complete listing of the routes is available on the web page.

5 Implementation and Experimental Results

One important feature of the layout strategy described in this paper, is its simplicity. An implementation of the layout is available in a package of 3-dimensional

drawing tools, called OrthoPak, from: `www.cs.uleth.ca/~wismath/packages`. This package was written in C++ and uses the LEDA library extensively. It runs under Solaris 2.6, or linux and produces a VRML world that can be examined with an appropriate browser. OrthoPak is free for research and teaching purposes and is part of a larger suite of research tools developed at the University of Lethbridge.

A test suite of graphs for 3-D orthogonal drawing was described in [3]. Our time trials indicate that our layout strategy is very efficient, 7.013 seconds for the entire test suite of 1820 graphs containing from 5 to 95 vertices running on a SPARC 5. On average, our layouts had between 4.71 and 4.83 bends per edge over the entire distribution of graphs. A more detailed account of the experiments is omitted for lack of space.

6 Conclusion and Open Problems

A 3-dimensional orthogonal layout of graphs with n vertices of maximum degree 6 was presented in which each edge is routed with at most 6 bends in a volume bounded by $O(n^2)$. The technique is fully dynamic allowing insertion and deletion of edges and vertices in $O(1)$ time. Multiple edges and self-loops are permitted. An implementation of the drawing strategy is provided, and VRML worlds and a short video describing the technique are also available at: `www.cs.uleth.ca/~wismath/threed`.

If the layout strategy is not used in a dynamic setting then some preprocessing of the graph is possible and the number of bends can be reduced in the average case. Alternately, special properties of the graph may also be exploitable to reduce the number of bends. For example, trees of maximum degree 6 can be embedded with our construction with no more than 5 bends. An interesting open problem is to determine if there are other classes of graphs that can be routed using this strategy with a minimal amount of pre-processing.

References

1. T. Biedl, Heuristics for 3D-Orthogonal Graph Drawings, *Proceedings of the 4th Twente Workshop on Graphs and Combinatorial Optimization*, Enshende, June 1995, pp. 41-44.
2. T. Biedl, T. Shermer, S. Whitesides, S. Wismath, Three-Dimensional Orthogonal Graph Drawing, *Symposium on Graph Drawing 97*, Lecture Notes in Computer Science **1353**, Springer Verlag, 1998, pp. 76-86.
3. G. Di Battista, M. Patrignani, F. Vargiu, A Split & Push Approach to 3D Orthogonal Drawing, *Symposium on Graph Drawing 98*, Lecture Notes in Computer Science **1547**, Springer Verlag, 1998. pp.87-101.
4. P. Eades, A. Symvonis, S. Whitesides, Two Algorithms for Three-Dimensional Orthogonal Graph Drawing, *Symposium on Graph Drawing 96*, Lecture Notes in Computer Science **1190**, Springer Verlag, 1996, pp. 139-154.
5. A. Papakostas, I. Tollis, Algorithms for Incremental Orthogonal Graph Drawing in Three Dimensions, *J. of Graph Algorithms and Applications*, to appear.

6. D. Wood, An Algorithm for Three-Dimensional Orthogonal Graph Drawing, *Symposium on Graph Drawing 98*, Lecture Notes in Computer Science **1547**, Springer Verlag, 1998. pp.332-346.

A Appendix: Routings

All 36 port-to-port routings from a vertex v to vertex w will now be enumerated. In each case, the initial and terminating points of the route are $(X(v), Y(v) = 0, Z(v))$ and $(X(w), Y(w) = 0, Z(w))$ respectively.

- $N \to S$: $(X(v), 1, Z(v)) \to (X(v), 1, Z(v)+2) \to (X(w)-1, 1, Z(v)+2) \to (X(w)-1, -2, Z(v)+2) \to (X(w), -2, Z(v)+2) \to (X(w), -2, Z(w))$
- $N \to N$: $(X(v), 1, Z(v)) \to (X(v), 1, Z(v)+2) \to (X(w), 1, Z(v)+2) \to (X(w), +2, Z(v)+2) \to (X(w), +2, Z(w))$
- $N \to E$: $(X(v), 1, Z(v)) \to (X(v), 1, Z(v)+2) \to (X(w)+2, 1, Z(v)+2) \to (X(w)+2, 0, Z(v)+2) \to (X(w)+2, 0, Z(w))$
- $N \to W$: $(X(v), 1, Z(v)) \to (X(v), 1, Z(v)+2) \to (X(w)-2, 1, Z(v)+2) \to (X(w)-2, 0, Z(v)+2) \to (X(w)-2, 0, Z(w))$
- $N \to T$: $(X(v), 1, Z(v)) \to (X(v), 1, Z(v)+2) \to (X(w)-3, 1, Z(v)+2) \to (X(w)-3, 0, Z(v)+2) \to (X(w)-3, 0, Z(w)+3) \to (X(w), 0, Z(w)+3)$
- $N \to B$: $(X(v), 1, Z(v)) \to (X(v), 1, Z(v)+2) \to (X(w), 1, Z(v)+2) \to (X(w), 0, Z(v)+2)$
- $S \to N$: $(X(v), -1, Z(v)) \to (X(v), -1, Z(v)+1) \to (X(w)-1, -1, Z(v)+1) \to (X(w)-1, 2, Z(v)+1) \to (X(w), 2, Z(v)+1) \to (X(w), 2, Z(w))$
- $S \to S$: $(X(v), -1, Z(v)) \to (X(v), -1, Z(v)+1) \to (X(w), -1, Z(v)+1) \to (X(w), -2, Z(v)+1) \to (X(w), -2, Z(w))$
- $S \to E$: $(X(v), -1, Z(v)) \to (X(v), -1, Z(v)+1) \to (X(w)+2, -1, Z(v)+1) \to (X(w)+2, 0, Z(v)+1) \to (X(w)+2, 0, Z(w))$
- $S \to W$: $(X(v), -1, Z(v)) \to (X(v), -1, Z(v)+1) \to (X(w)-2, -1, Z(v)+1) \to (X(w)-2, 0, Z(v)+1) \to (X(w)-2, 0, Z(w))$
- $S \to T$: $(X(v), -1, Z(v)) \to (X(v), -1, Z(v)+1) \to (X(w)-3, -1, Z(v)+1) \to (X(w)-3, 0, Z(v)+1) \to (X(w)-3, 0, Z(w)+3) \to (X(w), 0, Z(w)+3)$
- $S \to B$: $(X(v), -1, Z(v)) \to (X(v), -1, Z(v)+1) \to (X(w), -1, Z(v)+1) \to (X(w), 0, Z(v)+1)$
- $E \to N$: $(X(v)+1, 0, Z(v)) \to (X(v)+1, 1, Z(v)) \to (X(w), 1, Z(v)) \to (X(w), 2, Z(v)) \to (X(w), 2, Z(w))$
- $E \to S$: $(X(v)+1, 0, Z(v)) \to (X(v)+1, 1, Z(v)) \to (X(w)-1, 1, Z(v)) \to (X(w)-1, -2, Z(v)) \to (X(w), -2, Z(v)) \to (X(w), -2, Z(w))$
- $E \to E$: $(X(v)+1, 0, Z(v)) \to (X(v)+1, 1, Z(v)) \to (X(w)+2, 1, Z(v)) \to (X(w)+2, 0, Z(v)) \to (X(w)+2, 0, Z(w))$
- $E \to W$: $(X(v)+1, 0, Z(v)) \to (X(v)+1, 1, Z(v)) \to (X(w)-2, 1, Z(v)) \to (X(w)-2, 0, Z(v)) \to (X(w)-2, 0, Z(w))$
- $E \to T$: $(X(v)+1, 0, Z(v)) \to (X(v)+1, 1, Z(v)) \to (X(w)-3, 1, Z(v)) \to (X(w)-3, 0, Z(v)) \to (X(w)-3, 0, Z(w)+3) \to (X(w), 0, Z(w)+3)$
- $E \to B$: $(X(v)+1, 0, Z(v)) \to (X(v)+1, 1, Z(v)) \to (X(w), 1, Z(v)) \to (X(w), 0, Z(v))$

- $W \to N$: $(X(v)-1,0,Z(v)) \to (X(v)-1,3,Z(v)) \to (X(w),3,Z(v)) \to (X(w),2,Z(v)) \to (X(w),2,Z(w))$
- $W \to S$: $(X(v)-1,0,Z(v)) \to (X(v)-1,-3,Z(v)) \to (X(w),-3,Z(v)) \to (X(w),-2,Z(v)) \to (X(w),-2,Z(w))$
- $W \to W$: $(X(v)-1,0,Z(v)) \to (X(v)-1,-3,Z(v)) \to (X(w)-2,-3,Z(v)) \to (X(w)-2,0,Z(v)) \to (X(w)-2,0,Z(w))$
- $W \to E$: $(X(v)-1,0,Z(v)) \to (X(v)-1,-3,Z(v)) \to (X(w)+2,-3,Z(v)) \to (X(w)+2,0,Z(v)) \to (X(w)+2,0,Z(w))$
- $W \to T$: $(X(v)-1,0,Z(v)) \to (X(v)-1,-3,Z(v)) \to (X(w)-3,-3,Z(v)) \to (X(w)-3,0,Z(v)) \to (X(w)-3,0,Z(w)+3) \to (X(w),0,Z(w)+3)$
- $W \to B$: $(X(v)-1,0,Z(v)) \to (X(v)-1,-3,Z(v)) \to (X(w)+1,-3,Z(v)) \to (X(w)+1,0,Z(v)) \to (X(w),0,Z(v))$
- $T \to N$: $(X(v),0,Z(v)+3) \to (X(v),+1,Z(v)+3) \to (X(w),1,Z(v)+3) \to (X(w),2,Z(v)+3) \to (X(w),2,Z(w))$
- $T \to S$: $(X(v),0,Z(v)+3) \to (X(v),-1,Z(v)+3) \to (X(w),-1,Z(v)+3) \to (X(w),-2,Z(v)+3) \to (X(w),-2,Z(w))$
- $T \to E$: $(X(v),0,Z(v)+3) \to (X(v),+1,Z(v)+3) \to (X(w)+2,1,Z(v)+3) \to (X(w)+2,0,Z(v)+3) \to (X(w)+2,0,Z(w))$
- $T \to W$: $(X(v),0,Z(v)+3) \to (X(v),-1,Z(v)+3) \to (X(w)-2,-1,Z(v)+3) \to (X(w)-2,0,Z(v)+3) \to (X(w)-2,0,Z(w))$
- $T \to B$: $(X(v),0,Z(v)+3) \to (X(v),-1,Z(v)+3) \to (X(w),-1,Z(v)+3) \to (X(w),0,Z(v)+3)$
- $T \to T$: $(X(v),0,Z(v)+3) \to (X(v),+1,Z(v)+3) \to (X(w)-3,1,Z(v)+3) \to (X(w)-3,0,Z(v)+3) \to (X(w)-3,0,Z(w)+3) \to (X(w),0,Z(w)+3)$
- $B \to N$: $(X(v),0,Z(v)-1) \to (X(v),+1,Z(v)-1) \to (X(w),1,Z(v)-1) \to (X(w),2,Z(v)-1) \to (X(w),2,Z(w))$
- $B \to S$: $(X(v),0,Z(v)-1) \to (X(v),-1,Z(v)-1) \to (X(w),-1,Z(v)-1) \to (X(w),-2,Z(v)-1) \to (X(w),-2,Z(w))$
- $B \to E$: $(X(v),0,Z(v)-1) \to (X(v),+1,Z(v)-1) \to (X(w)+2,1,Z(v)-1) \to (X(w)+2,0,Z(v)-1) \to (X(w)+2,0,Z(w))$
- $B \to W$: $(X(v),0,Z(v)-1) \to (X(v),-1,Z(v)-1) \to (X(w)-2,-1,Z(v)-1) \to (X(w)-2,0,Z(v)-1) \to (X(w)-2,0,Z(w))$
- $B \to T$: $(X(v),0,Z(v)-1) \to (X(v),+1,Z(v)-1) \to (X(w)-3,1,Z(v)-1) \to (X(w)-3,0,Z(v)-1) \to (X(w)-3,0,Z(w)+3) \to (X(w),0,Z(w)+3)$
- $B \to B$: $(X(v),0,Z(v)-1) \to (X(v),-1,Z(v)-1) \to (X(w),-1,Z(v)-1) \to (X(w),0,Z(v)-1)$

An $E \log E$ Line Crossing Algorithm for Levelled Graphs

Vance Waddle and Ashok Malhotra

IBM Thomas J. Watson Research Center,
P.O.Box 704
Yorktown Heights, NY 10598
{waddle,petsa}@us.ibm.com

Abstract. Counting the number of crossings between straightline segments is an important problem in several areas of Computer Science. It is also a performance bottleneck for Sugiyama-style layout algorithms. This paper describes an algorithm for leveled graphs, based on the classification of edges that is $O(e \log e)$ where e is the number of edges. This improves on the best algorithm in the literature which is $O(e^{1.695} \log e)$. The improved crossing algorithm enabled an implementation of a Sugiyama-style algorithm to lay out graphs of tens of thousands of nodes in a few seconds on current hardware.

1 Introduction

The design of algorithms to compute the intersections of straight line segments, and counting their intersections, is a well known problem in computational geometry. A classic use for these algorithms in computer graphics is to determine a line's intersection with a viewing area's "clipping" bounds. There can be e^2 intersections on e line segments, so an algorithm that examines intersections is $O(e^2)$.

A more recent area concerned with line intersections is graph layout for user interfaces [4, 6, 12, 14]. These algorithms lay out a directed graph according to various visual aesthetics to improve the graph's readability as part of a user interface displaying, for example, a flow-chart, a class hierarchy, or a database schema. Two of the most common aesthetics are 1) *leveled graph* - display the nodes in the graph in levels, where all the nodes on a level have the same vertical coordinate and 2) reduce the number of edge intersections to make it easier to visually follow the edges between nodes. The second aesthetic implies that counting the number of line intersections is a fundamental operation in these layout algorithms, since they must compare the number of edge intersections in alternative layouts to determine the better layout. Worse, since Sugiyama-style crossing reduction algorithms [14] are based on sorting, these algorithms could be $O(n \log n)$ in the number of nodes, except for the intersection count. We have discovered by experience that counting line intersections is a performance bottleneck for the entire algorithm for large graphs.

J. Kratochvíl (Ed.): GD'99, LNCS 1731, pp. 59–71, 1999.

Sugiyama-style crossing reduction algorithms perform well for "small" graphs of up to a few hundred nodes, even with a simple $O(e^2)$ algorithm for counting the edge crossings. However, recent work [8, 10] has begun to consider the problem of handling graphs of tens of thousands, or even hundreds of thousands of nodes. Munzer [10] argues that more general, non-tree layout algorithms are too slow to process this class of graphs.

In the remainder of the paper, we first discuss previous algorithms, and their relationship to our work. We then describe the layout problems which motivate our algorithm, and show how to compute the number of intersections in a leveled graph by using the graph's geometry. Next, we describe our line intersection algorithm, and prove bounds on its worse case performance. We describe its performance when implemented and run on a sequence of test cases and, finally, give our conclusions.

2 Previous Work

The simplest algorithm for determining the number of intersections of a collection of line segments is to test each pair of line segments for intersection. Where e is the number of line segments, this algorithm is $O(e^2)$. While there can be e^2 intersections in the worst case, there are usually far fewer intersections than this, leaving this algorithm a poor solution in practice. When implemented on a leveled graph, only line segments between adjacent levels are tested for intersection. This algorithm works better than might be expected for small graphs, both because the intersection test is cheap (comparing the positions of the end nodes), and because the nodes tend to be spread among the levels, which divides the problem into smaller pieces.

The main cost of the simple crossing algorithm lies in repeatedly testing segments for intersections as they occur in different segment pairs. Sweep algorithms [1, 3] avoid re-examining edges by performing a single sweep across the plane. Segments are tested for intersection only when they overlap the sweep line. These algorithms list the intersections of straight line segments, and are $O(e \log e + k)$, where e is the number of line segments and k the number of intersections. They are $O(e^2)$ in the number of edges, since there can be that many intersections in the worst case. The best reported algorithm for just counting intersections [2] is order $O(e^{1.695} \log e)$ and is based on an algorithm for partitioning the plane into regions by [5].

Graph layout algorithms only need a count of the number of intersections, not a list of the intersecting line segments. Thus, they do not require examining individual pairs of line segments to determine intersections. In leveled graphs, the levels decompose the graph into disjoint regions (although not the regions required by [2]), so the crossing algorithm does not require the overhead of decomposing the graph into regions if it utilizes the level structure properly. By using the geometry of a leveled graph to classify edges based on their relationship to the current sweep position, we create a simple algorithm to compute the number of line intersections that is $O(e \log e)$, where e is the number of edges.

3 Leveled Graphs and Layout

In this section we briefly describe leveled graphs and the relationship between leveled graphs and the layout algorithms that create them that motivate our intersection algorithm.

A *leveled graph* is a directed graph in which each node in the graph has been assigned to a single *level*, an ordered set of nodes, from an ordered set of levels. When the level is displayed horizontally, all the nodes in the level have the same vertical coordinate. Further, the end nodes of each edge are placed on adjacent levels in the graph, so that each edge spans a single level. If, after nodes are assigned to levels, there are ”long edges” for which the source and target end nodes are not on adjacent levels, these edges are ”shortened” by adding ”virtual” intermediate nodes so that each edge in the resulting graph spans a single pair of adjacent levels. For a complete description of leveled graphs and their layout algorithms, see [4].

Figure 1 shows an example leveled graph consisting of nodes **A** through **H**. Node **A** is on the first (top) level, **B** and **C** on the second level, etc. (The graph is displayed as it might be after the layout process is completed, and coordinates have been assigned to all the nodes in the graph.) Virtual nodes have been added between nodes **A** and **D**, and nodes **C** and **F**. Notice the paths for nodes **C** and **H**, and **D** and **G** allow some freedom in their assignment to a level. We emphasized this in the figure by placing **C** and **H** on the highest possible level, and **D** and **G** on the lowest.

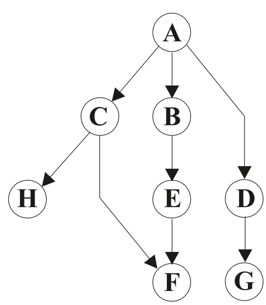


Figure 1. A leveled graph

Sugiyama-style layout algorithms [14] separate the layout problem into three phases. The first phase produces a leveled graph in which each node is assigned to a level, with child nodes appearing on a level beneath their parent. A typical scheme to assign nodes to levels is to use a variation of a topological sort. Virtual nodes are added to shorten long edges during this phase. This procedure only partially constrains the assignment of nodes to levels. Some algorithms add additional processing to improve the distribution of nodes to levels. For example,

the DAG system [6] uses integer programming to reduce total edge lengths in the graph.

The second phase re-arranges nodes within their assigned levels to reduce the number of crossings. The algorithms [4] use heuristics, typically based on sorting, to reduce the number of crossings, since it is known that minimizing intersections is an NP complete problem [7]. It is during this phase that counting the number of intersections becomes a critical problem. After crossings have been reduced by the second phase, the nodes have a relative position within the level, but no coordinates. The third phase is a positioning phase that assigns coordinates to levels, and spaces the nodes within their levels.

4 Leveled Graphs and Intersections

In this section we describe how to efficiently compute the number of edge intersections in a leveled graph. We show how to classify edges into types based on the current sweep position and use the classification to compute the number of edge intersections. We then prove that all intersections are captured by our procedure.

Our algorithm makes a single sweep across a level, and uses the geometry of a leveled graph to remove the test for intersection by classifying the edges into types based on the position of their end-nodes with respect to the current sweep position. The number of intersections is then computed using the geometry of the leveled graph, and the position of the edge's end nodes. It follows from the geometry of the graph and the definition of the edge types that the edges of certain types must intersect. We then prove that all intersections are counted at some stage of the sweep via this classification.

Our discussion focuses on a pair of adjacent levels, since all intersections in a leveled graph appear between adjacent levels. (Note: unlike the classical segment crossing problem in computational geometry, edges with a common end node do not count as an intersection). We assume that nodes are numbered on each level, from left to right, starting with 1. We denote an edge e between a node a in the top level and b in the bottom level as the tuple $<a,b>$. In the following, when we compare nodes with arithmetic equalities and inequalities we are comparing the nodes' ordinal positions within the level. Also, note that since the algorithm computes intersections by making a sweep (from left to right) of the nodes in the pair of levels being examined, the sweep position is always a pair of nodes $<a,b>$ with equal positions, i.e., $a = b$.

Definition of edge types: An edge between nodes $<P_i,\ P_j>$ is classified as being a *top level* edge if $P_i < P_j$, a *bottom level* edge if $P_i > P_j$, and *pair edge* if $P_i = P_j$. If the current sweep position is S, each top-level and bottom-level edge is further classified as

1) *Right* edge if $P_i = S$, and $P_i < P_j$, or if $P_j = S$, and $P_j < P_i$,

2) *Trailing* edge if $P_i < S$ and $S < P_j$, or if $P_j < S$ and $S < P_i$.

Note that an edge's type is a function of the current sweep position, and changes along with the sweep position.

We illustrate the edge types through the leveled graph shown in Figure 2. Nodes in the top level are nodes a through e, and the bottom level nodes are u through z. Suppose the sweep is at the pair of nodes $<c,y>$. Top level edges are: $<c,y>$ is the graph's single pair edge, $<c,v>$ and $<c,u>$ are right edges, and $<a,v>$, $<a,u>$, $<b,v>$, and $<b,u>$ are trailing edges. Bottom level edges $<y,d>$ and $<y,e>$ are right edges, and $<z,d>$, $<z,e>$, $<x,d>$, and $<x,e>$ are trailing edges.

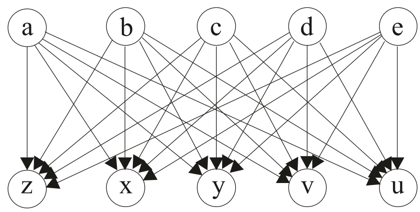


Figure 2. Adjacent levels in graph.

Given the geometry, it is clear that top level trailing edges must cross with bottom right edges, that top and bottom level right edges must cross, etc. The number of these crossings can be calculated simply from knowing the numbers of each type of edge. Let R_T be the number of top level right edges, R_B be the number of bottom level right edges, P the number of pair edges, Tr_T the number of top trailing edges, etc., the number of crossings due to these sources is

$$Tr_T * R_B + Tr_B * (R_T + P) + R_T * R_B$$

We call these crossings *alternate level crossings* because they are produced by edges from the top level intersecting with edges from the bottom level. Unfortunately, there is an additional source of crossings shown in our example graph. Right edges and trailing edges from the same level can intersect, as shown by the intersection of $<c,v>$ with $<a,u>$ and $<b,u>$. We call these *same level crossings.* These intersections are more difficult to determine than our edge types, since not all right and trailing edges from the same level cross, e.g., $<c,u>$ and $<a,u>$ do not cross.

By examining the geometry, we note that a same-level crossing occurs between a right top level edge, $<t_1,b_1>$, and a top level trailing edge, $<t_2,b_2>$, iff $b_1 < b_2$. So the number of crossings due to a right edge $<t_1,b_1>$ is the number of trailing edges to the nodes at positions $b_1 + 1$ through b_{LAST}, where b_{LAST} is the position of the last node on the bottom level. Similar considerations hold for same-level crossings from the bottom level. Note that we count crossings between pair edges and top trailing edges as same-level crossings.

If we let $Same_T$ and $Same_B$ be the number of top and bottom same level crossings at a sweep position, then the number of crossings at a given sweep

position is

$$Tr_T * R_B + Tr_B * (R_T + P) + R_T * R_B + Same_T + Same_B$$

This leaves us with two problems: 1) proving that the above expression covers all the intersections, and 2) finding an efficient scheme for answering the range query required to determine the number of same level intersections due to a given right edge.

We prove that all intersections are captured by starting with an intersection between two edges, and working backward to show at which stage of the sweep it was considered, and which type of intersection under which it was classified. The proof follows by simply elaborating the cases under the trichotomy law.

Proof: All intersections are captured either as an intersection between different edge types, or as a same-level intersection between a level's right and trailing edges. Consider an intersection between distinct edges *e1* to nodes $<a,b>$ and *e2* to nodes $<c,d>$.

- Cases for $a < b$: If $a < b$, then $e1$ is a top edge.
  - $c > d$: In this case, $e2$ is a bottom edge. If $c \leq a$, then $e1$ and $e2$ do not intersect. If $c > a$ and there is an intersection, then it must be that $d < b$. If $a = d$ then $e1$ and $e2$ intersect as right edges (covered); if $a < d$ then $e1$ is a top trailing edge and the intersection occurs when $e2$ is a bottom right edge (covered); if $a > d$ then e1 is right edge and the intersection occurs when $e2$ is a trailing edge (covered).
  - $c = d$: Edge $e2$ is a pair edge. If $a < c$ and $c < b$, then $e1$ is a trailing edge, and intersects $e2$ when the sweep position is c (covered). If $a \leq c$ or $c \geq b$, there is no intersection.
  - $c < d$: Edge $e2$ is a top edge, and any intersection occurs as a same level intersection between $e1$ and $e2$. For an intersection to occur, either $a < c$ and $b > d$ or $a > c$ and $b < d$. If $a < c$ and $b > d$, then $e1$ is a trailing edge, and intersects with $e2$ when c becomes the sweep position, with $e2$ a right edge (covered). Similarly, if $a > c$ and $b < d$, then $e1$ is a right edge when a becomes the sweep position, and $e2$ is a trailing edge (covered).
- Cases for $a = b$: If $a = b$, then $e1$ is a pair edge.
  - $c > d$: $e2$ is a bottom edge. If $c \leq a$, or $b \leq d$, there is no intersection. If $c > a$ and $d < b$, then $e1$ intersects $e2$ when the sweep position is a, and e2 is a trailing edge (covered).
  - $c = d$: Edge $e1 = e2$, and since they were assumed to be distinct edges, this does not occur.
  - $c < d$: If $c \geq a$ or $d \geq b$, there is no intersection. If $c < a$, then $e2$ intersects $e1$ as a trailing edge (covered).
- Cases for $a > b$. Here, $e1$ is a bottom edge.
  - $c > d$: $e2$ is also a bottom edge. The cases here are the same as when $a < b$ and $c < d$, and both were top edges.

– $c = d$: $e2$ is a pair edge. If $a \geq c$ or $b \geq d$, there is no intersection. If $a < c$, and $b > d$, then the edges intersect with $e1$ a trailing edge (covered).
– $c < d$: $e2$ is now a top edge, and the cases follow similarly to when the roles of $e1$ and $e2$ were reversed (when $a < b$ and $c > d$).

End proof.

We introduce a structure we call an *accumulator tree* to efficiently compute the number of trailing edges to a range of nodes. There are two trees, one for the top level and one for the bottom level. Each leaf in the binary tree is the current count of trailing edges into a node of the given level. Non-leaf nodes contain the sum of the counts in their children. The number of trailing edges to a range of nodes $N_{j,\ldots,}N_{LAST}$ is computed by walking up the tree from N_j, and summing the counts of the left subtrees into the path from the node to tree's root, and subtracting it from the value in the tree's root. In effect, we keep a sum of all the ranges in the tree, and then subtract out the portions in which we are not interested.

The accumulator tree is implemented by mapping it into an array, with the tree's root node at position 1, its left and right children at positions 2 and 3, etc. [9]. Thus, if the sequence of positions from a leaf node to the root is $P_{1,\ldots,}P_k$, then if P_j is odd, we add the value of the entry in the tree for P_{j-1}. Figure 3 illustrates this for a level with 8 nodes, and shows the path through the tree to query the count into nodes at positions 6 through 8. The bracketed number is the position of a node in the array, and the unbracketed number is the position number of the node within the level. The square nodes are the path from node 6 (at entry "[13]") to the root node, and the nodes with a dot (nodes at "[2]" and "[12]") are nodes whose counts are summed to remove from the root node at "[1]".

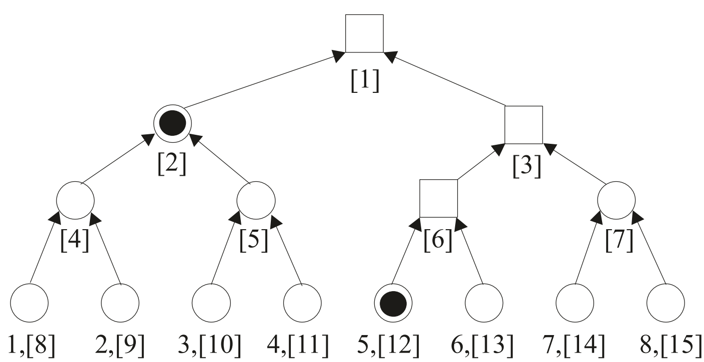


Figure 3. Accumulator tree.

The three operations of interest on the accumulator tree are: 1) CountEdges - compute the number of edges in a range of nodes (our range query), 2) AddEdges - add an edge to the tree, and 3) RemoveEdge - remove an edge from the tree.

We give below the procedure for CountEdges. The argument Tree gives the array implementing the accumulator tree, and n is the end node for the edge being counted. AddEdge adds one to each node on a path from from a leaf node to the root node. RemoveEdge is identical to AddEdge, except it decrements the count by one. "Base" is the amount added to map the node at position 1 into its position in the tree's array.

```
CountEdges(Tree, n, last) {
    if ( n == last )
      return 0;
    Pos = n + Base + 1;
    Sum = 0;
    while( Pos >= 1) {
       if ( Odd(Pos) )
         Sum = Sum + Tree[Pos - 1]
       Pos = Pos / 2;
    }
    return( Tree[1] - Sum);
}
```

Notice that Tree[1] is the sum of all the trailing edges for the current sweep position.

5 The LevelCross Algorithm

We can now give our algorithm to count crossings in a leveled graph, which we call the LevelCross algorithm. In the following, "AccL" and "AccM" are the arrays for the accumulator trees for levels L and M. "Count" is the number of crossings, "Pair" the number of pair edges, and "Same" the number of same level edges. Rt and Rb are the number of right-top and right-bottom edges, respectively, and Trt and Trb the number of trailing top and bottom edges. The predicate "IsPair(e)" is true iff e is a pair edge, and the predicate "Right(e)" is true iff e is a right edge.

For each level in the graph, the algorithm makes a sweep over the level, counting the edges of each type, based on the current sweep position. After the number of edge intersections is computed for the sweep position, the accumulator trees and counts are updated to reflect the change to the next sweep position.

LevelCross:
Count = 0;
for each level L of nodes $\{L_1, ..., L_K\}$
x with a successor level M of nodes $\{M_1, ..., M_J\}$
initialize trees AccL and AccM, for levels L and M, respectively, to 0
for I = 1 to $\min(K, J)$ // sweep position is $< L_I, M_I >$
Pair = Rt = Rb = Same = 0;
for each edge e = $< L_I, x >$ from L_I // Count top edges

```
            if ( Right(e)) {
                Rt = Rt + 1
            } else if ( IsPair(e)) {
                Pair = Pair + 1
            }
            Same = Same + CountEdges( AccM, x, J)
    }
    Trt = AccL[1]
    Trb = AccM[1]
    for each edge e = < x, M_I > from M_I  // Count bottom edges
        if ( Right(e))
            Rb = Rb + 1
            Same = Same + CountEdges( AccL, x, K)
        }
    }
    Count = Count + Trt*Rb + Trb*(Rt + Pair) + Rt*Rb + Same
    Move Sweep Position:

        1) Remove trailing edges to L_I and M_I from their respective trees.
        2) Right edges from L_I become trailing edges in bottom tree
            (and are added to the tree), and similarly for right
            edges from M_I in top tree.
    }
}
```

The intersection counting algorithm runs in O(e log e) time:

Proof:
The number of nodes on a level ranges between 1 and e, so each pass through a level's accumulator tree requires log e time, so within a loop over edges it requires *O(e log e)*. Based on simply the number of loops over edges, the algorithm might appear to run in time $6e$ log e. However, it really requires only $3e$ log e since each edge is traversed once, as either a top or bottom edge. Further, the "sweep" loop over the nodes in a level does not contribute another multiple of "e" to the running time. This is because the nodes "partition" the edges in the sense that an edge is only examined via its end nodes, so edges are not repeatedly examined for each node. In effect, the nodes are used as a mechanism to iterate over the edges.

Thus, the algorithm's running time is *O(e log e)*.
End Proof.

6 Implementation Results

This section describes the performance measured when the LevelCross algorithm was implemented as part of the layout code in the NARC (Nodes and ARC) graph toolkit, which is an internal IBM tool that provides a high-level service to display, layout, and edit graphs as part of a user interface. LevelCross

was implemented to replace the previous intersection counting algorithm in NARC's Sugiyama-style layout algorithm. It replaced the naive algorithm of simply sweeping over the nodes in the level and counting the crossings. The measurements were done on a 200 MHz Pentium Pro running OS/2 Warp (4.0). Notice that this is a much harsher test than just the performance of the intersection algorithm. The intersection algorithm must both be faster than the simpler algorithm, *and* counting intersections must be a bottleneck to the entire layout process. Otherwise, improvements due to the intersection algorithm will be hidden by the time required to perform the rest of the layout algorithm.

We devised an algorithm for generating random leveled graphs to produce the graphs used as input data. This allowed us to create graphs with controlled sizes and properties on demand. Briefly, the input to the algorithm consists of a set of input "tiles" (subgraphs), a desired graph size (number of nodes), the desired aspect ratio (number of levels divided by number of nodes per level), and the specification of "generators" to randomly select properties such as in-degree and the length of edges used to connect nodes. The nodes in each tile have a pre-computed level structure, and the tile has a set of its nodes on its top level designated as input nodes, and a set of nodes on its bottom level as output nodes. Each row of tiles is filled by creating instances of randomly selected tiles until the row had the desired width. The input nodes in each tile are connected to a randomly selected <tile, output node> pair from a preceding row of tiles.

We measured the execution time in milliseconds of the layout algorithm with the naive, "Simple" $O(e^2)$ crossing algorithm, and for the LevelCross algorithm on two series of randomly generated graphs. In each series of graphs, the number of target nodes in each graph were doubled from the previous graph. In the first series (shown in Table 1), the graphs were generated from a library of tiles in which each tile had 4 levels, and the in-edges were connected to a previous row selected from a uniformly distributed range of 1 to 4. (A "1" connects to a tile in the previous row, "2" the row before that, etc. This process creates long edges acording to the chosen distribution.)

As shown in Table 1, the graphs ranged in size from 168 nodes and 281 edges to 40,444 nodes and 74,189 edges in the input graph. The columns "Lay nodes" and "Lay edges" give the internal size of the graph after long edges have been replaced by a sequence of edges to virtual nodes, and the "density" columns give the ratio of edges/nodes for their respective columns. Initially, the layout times are the same for both crossing algorithms, but diverge with increasing graph sizes. (A 40K node graph was as large as the 64 Meg. memory on our test machine could handle.) On all but the last graph, the crossing reduction phase accounted for between 92 and 95 percent of the total layout time when using the simple algoritm, and accounted for 77 percent of the total layout time when level cross was implemented. In the last graph, the cost of creating the level structure and long edges increased by an order of magnitude. Since the number of nodes and edges only doubled, we assume this is due to exhausting memory.

To explore the effect of increased edge density, we generated a second series of graphs based on tiles with two levels, raised the in-degree of the nodes, and prevented generation of long edges, with the results shown in Table 2. (Both

Table 1

In Nodes	In edges	Lay nodes	Lay edges	In density	Lay density	Simple time(ms)	LevelCross time(ms)
168	281	261	374	1.67	1.43	16	16
341	569	640	868	1.67	1.36	47	39
680	1,193	1,509	2,002	1.75	1.34	148	109
1,297	2,314	3,340	3,341	1.78	1.3	414	250
2,611	4,710	7,857	9,956	1.8	1.27	1,383	625
5,121	9,350	16,945	21,174	1.83	1.25	3,961	1,344
10,174	18,713	33,065	41,064	1.79	1.24	10,039	2,672
20,117	37,001	69,654	86,538	1.84	1.24	29,219	5,914
40,444	74,189	139,220	172,965	1.83	1.24	69,024	15,563

Table 2

In Nodes	In edges	Density	Simple time(ms)	LevelCross time(ms)
157	499	3.18	23	16
312	1,145	3.27	31	23
633	2,191	3.46	94	54
1,275	4,387	3.4	218	109
2,524	8,869	3.51	734	274
5,076	18,138	3.57	2,000	586
10,044	35,485	3.53	5,375	1,242
20,003	71,885	3.59	15,000	2,477
40,081	143,806	3.59	42,063	5,632

long edges and tiles with more levels limit the edge density.) Since these graphs have no long edges, there are the same number of edges in the layout as in the input graph, and we omit separate columns for the layout graph. If we compare the results inTable 2 with those in Table 1, the simple algorithm degrades faster with a higher edge density. For example, for the last graph in Table 2, the layout with the simple algorithm takes over 7 times as long as LevelCross, where it only took 4.4 times as long for 40K node input graph in Table 1. The times are longer in Table 1 because the size of the layout graph is larger due to long edges.

These results show that the line crossing algorithm is a performance bottleneck for Sugiyama-style layout algorithms for large graphs, and our algorithm substantially improves its performance. Beyond this, we feel there are some lessons beyond a better crossing algorithm. First, Sugiyama-style algorithms are fast enough to lay out graphs of tens, or even hundreds of thousands of nodes. The 200 MHz Pentium Pro machine is now several years old, and ran out of memory, or we could have experimented with larger graphs. Thus, it is not necessary to restrict the layout algorithm to a spanning tree as was done in [Munzer] to handle very large graphs. Second, we found to our surprise that graphs of tens of thousands of nodes were "readable" on a 1600 by 1200 pixel display in the

sense we could still see individual nodes and edges. The commonly held "folk theorem" seems to be that it is pointless to display graphs of more than a few hundred nodes without (possibly heavy) filtering, because they would be completely unreadable. In fact, we discovered that if the graph is not too dense, and the nodes not distributed too unevenly on the screen, very large graphs can be viewed with current technology. Further, display technology is improving, as exemplified by IBM's recent announcement [13] of a prototype LCD display with a resolution of 2048 by 2560 pixels. This display has almost 3 times the number of pixels in our display, and could usefully display even larger graphs.

Finally, we feel the entire area of graph drawing would benefit from the development of algorithms to generate graphs on demand for testing layout algorithms. While it is always better to have real-life examples rich with interesting semantics, these graphs are often valuable because they are rare. They may also not have the desired size or other properties needed to exercise a particular aspect of a layout algorithm. We found it very convenient to generate graphs by specifying some descriptive parameters, and believe this is a promising area for future research.

7 Conclusions

We have described an $O(e \log e)$ algorithm for counting the number of edge crossings in a leveled graph, and verified its performance by implementation. We have also shown that the line crossing algorithm can be a performance bottleneck in Sugiyama-style algorithms, and the new algorithm significantly improves their performance. The algorithm achieves its performance by taking advantage of the geometry of leveled graphs to classify the edges in the graph into types based their relationship to the current sweep position, and from a scheme to efficiently compute range queries on the end nodes of edges.

References

1. Bentley, J.L. and Ottman, T. Algorithms for reporting and counting geometric intersections, *IEEE Trans. Comput.* C-28, (1979), pp 643-647.
2. Chazelle, B. Reporting and Counting Segment Intersections, *Journal of Computer and System Sciences*, Vol. 32, pp. 156-182, 1986.
3. Chazelle, B., and Edelsbrunner, H. An Optimal Algorithm for Intersecting Line Segments in the Plane, *JACM*, 39(1), pp. 1-54, Jan 1992.
4. Di Battista, G., Eades, P., Tamassia, R., and Tollis, I.G. *Graph Drawing: Algorithms for the Visualization of Graphs*, Prentice Hall, 1999.
5. Edelsbrunner, H., and Welzl, E. Halfplanar Range Search in Linear Space and $O(n^{0.695})$ Query Time," *Tech. Univ. Graz*, IIG Report 111, 1983.
6. Gansner, E. R., North, S. C., and Ko, K. P. DAG: A program that draws directed graphs, *Software Practice and Experience*, 18(11), pp. 1047-1062, Nov 1988.
7. Garey, M. R., and Johnson, D.S. Crossing number is NP-Complete, *SIAM Journal on Algebraic and Discrete Methods*, 4, 3 (September 1983), pp. 312-316.

8. Huang, M. L., Eades, P., and Wang, J., On-line Animated Visualization of Huge Graphs using a Modified Spring Algorithm, *Journal of Visual Languages and Computing*, Vol 9., pp. 623-645 (1998).
9. Knuth, D. E., *The Art of Computer Programming, Vol. 1: Fundamental Algorithms*, Second Ed., pp. 400-401, Addison-Wesley, 1973.
10. Munzer, T., Exploring Large Graphs in 3D Hyperbolic Space, *IEEE Computer Graphics and Applications*, Vol. 18 No. 4, 1998.
11. Preparata, F. P., and Shamos, M. I., *Computational geometry - An introduction.* Springer-Verlag, New York, 1985.
12. Rowe, L., et. al., A Browser for Directed Graphs, *Software Practice and Experience*, Vol. 17(1), pp. 61-76, Jan. 1987.
13. Schleupen, K., P. Alt, et. al., "High Information Content Color 16.3 inch Desktop AMLCD with 15.7 Million a-Si:H TFTs," *Asia Display* '98, Digest, Seoul Korea, (1998) pp. 187-191.
14. Sugiyama, K., Tagawa, S., and M. Toda, Methods for Visual Understanding of Hierarchical Structures, *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. SMC-11, No. 2. Feb. 1981.

Level Planar Embedding in Linear Time*

Michael Jünger and Sebastian Leipert

Institut für Informatik, Universität zu Köln, 50969 Köln, Germany,
{mjuenger,leipert}@informatik.uni-koeln.de

Abstract. In a level directed acyclic graph $G = (V, E)$ the vertex set V is partitioned into $k \leq |V|$ levels $V^1, V^2, \ldots, V^k$ such that for each edge $(u, v) \in E$ with $u \in V^i$ and $v \in V^j$ we have $i < j$. The level planarity testing problem is to decide if G can be drawn in the plane such that for each level V^i, all $v \in V^i$ are drawn on the line $l_i = \{(x, k - i) \mid x \in \mathbb{R}\}$, the edges are drawn monotonically with respect to the vertical direction, and no edges intersect except at their end vertices.
In order to draw a level planar graph without edge crossings, a level planar embedding of the level graph has to be computed. Level planar embeddings are characterized by linear orderings of the vertices in each V^i $(1 \leq i \leq k)$. We present an $\mathcal{O}(|V|)$ time algorithm for embedding level planar graphs. This approach is based on a level planarity test by Jünger, Leipert, and Mutzel [6].

1 Introduction

A fundamental issue in Automatic Graph Drawing is to display hierarchical network structures as they appear in software engineering, project management and database design. The network is transformed into a directed acyclic graph that has to be drawn with edges that are strictly monotone with respect to the vertical direction. Many applications imply a partition of the vertices into levels that have to be visualized by placing the vertices belonging to the same level on a horizontal line. The corresponding graphs are called level graphs. Using the PQ-tree data structure, Jünger et al. [6] have given an algorithm that tests in linear time whether such a graph is level planar, i.e. can be drawn without edge crossings.

In order to draw a level planar graph without edge crossings, a level planar embedding of the level graph has to be computed. Level planar embeddings are characterized by linear orderings of the vertices in each V^i $(1 \leq i \leq k)$. We present a linear time algorithm for embedding level planar graphs. Our approach is based on the level planarity test and augments a level planar graph G to an st-graph G_{st}, a graph with a single sink and a single source, without destroying the level planarity. Once the st-graph has been constructed, we compute a planar embedding of the st-graph. This is done by applying the embedding algorithm of Chiba et al. [2] for general graphs, obeying the topological ordering of the

* Supported by DFG-Grant Ju204/7-3, Forschungsschwerpunkt "Effiziente Algorithmen für diskrete Probleme und ihre Anwendungen"

J. Kratochvíl (Ed.): GD'99, LNCS 1731, pp. 72–81, 1999.

vertices in the st-graph. Exploiting the embedding of the st-graph G_{st}, we are able to determine a level planar embedding of G.

This extended abstract is organized as follows. After summarizing the necessary preliminaries in the next section, including a short introduction to the PQ-tree data structure and the level planarity test presented by Jünger et al. [6], we present in the third section the concept of the linear time level planar embedding algorithm. The fourth section concentrates on some details concerning the embedding algorithm. We close with some remarks on how to produce a level planar drawing using the results of our algorithm.

2 Preliminaries

Let $G = (V, E)$ be a directed acyclic graph. A *leveling* of G is a topological numbering $\text{lev} : V \to \mathbb{Z}$ mapping the vertices of G to integers such that $\text{lev}(v) \geq \text{lev}(u) + 1$ for all $(u, v) \in E$. G is called a *level graph* if it has a leveling. If $\text{lev}(v) = j$, then v is a *level-j vertex*. Let $V^j = \text{lev}^{-1}(j)$ denote the set of level-j vertices. Each V^j is a *level* of G. If $G = (V, E)$ has a leveling with k being the largest integer such that V^k is not empty, G is said to be a *k-level graph*. For a k-level graph G, we sometimes write $G = (V^1, V^2, \dots, V^k; E)$. A level graph $G = (V, E)$ is said to be *proper* if every edge $e \in E$ connects only vertices belonging to consecutive levels. A *hierarchy* is a level graph such that all sources belong to the first level V^1.

A drawing of G in the plane is a *level drawing* if the vertices of every V^j, $1 \leq j \leq k$, are placed on a horizontal line $l_j = \{(x, k - j) \mid x \in \mathbb{R}\}$, and every edge $(u, v) \in E$, $u \in V^i$, $v \in V^j$, $1 \leq i < j \leq k$, is drawn as a monotone curve between the lines l_i and l_j. A level drawing of G is called *level planar* if no two edges cross except at common endpoints. A level graph is *level planar* if it has a level planar drawing. A level graph G is obviously level planar if and only if all its components are level planar.

A level drawing of G determines for every V^j, $1 \leq j \leq k$, a total order $\leq_j$ of the vertices of V^j, given by the left to right order of the vertices on l_j. A *level embedding* consists of a permutation of the vertices of V^j for every $j \in \{1, 2, \dots, k\}$ with respect to a level drawing. A level embedding with respect to a level planar drawing is called *level planar*.

A PQ-tree is a data structure that represents the permutations of a finite set U in which the members of specified subsets occur consecutively. This data structure has been introduced by Booth and Lueker [1] to solve the problem of testing for the consecutive ones property. A PQ-tree contains three types of nodes: leaves, P-nodes, and Q-nodes. The leaves are in one to one correspondence with the elements of U. The P- and Q-nodes are internal nodes. A P-node is allowed to permute its children arbitrarily, while the order of the children of a Q-node is fixed and may only be reversed. In subsequent figures, P-nodes are drawn as circles while Q-nodes are drawn as rectangles.

The set of leaves of a PQ-tree T read from left to right is denoted by $\text{frontier}(T)$ and yields a permutation on the elements of the set U. The frontier

of a node X, denoted by frontier(X), is the sequence of its descendant leaves. Given a PQ-tree T over the set U and given a subset $S \subseteq U$, Booth and Lueker [1] developed a pattern matching algorithm called *reduction* and denoted by REDUCE(T, S) that computes a PQ-tree T' representing all permutations of T in which the elements of S form a consecutive sequence.

Let G^j denote the subgraph of G induced by $V^1 \cup V^2 \cup \cdots \cup V^j$. The strategy of testing the level planarity is to perform a top-down sweep, processing the levels in the order $V^1, V^2, \ldots, V^k$ and computing for every level V^j, and every component of G^k the set of permutations of the vertices of V^j that appear in some level planar embedding of G^j. Obviously, the graph $G = G^k$ is level planar, if and only if the set of permutations for G^k is not empty. This test implies for every level planar component of G^j that the set of its level planar embeddings can be represented by a PQ-tree.

As long as different components of G^j are not adjacent to a common vertex on level $j+1$, standard PQ-tree techniques using the function REDUCE are applied for constructing the PQ-tree of every component. If two or more components are adjacent to a common vertex v on level $j+1$, they have to be "merged" and a new PQ-tree has to be constructed from the two corresponding PQ-trees.

The merge operation is accomplished using certain information that is stored at the nodes of the PQ-trees. For any subset S of the set of vertices in V^j, $1 \leq j \leq k$, that belong to a component R_i^j (the i^{th} component of G^j), define ML(S) to be the greatest $d \leq j$ such that $V^d, V^{d+1}, \ldots, V^j$ induces a subgraph in which all vertices of S occur in the same connected component. The level ML(S) is said to be the *meet level* of S. For a Q-node Y in the corresponding PQ-tree $T(R_i^j)$ with ordered children $Y_1, Y_2, \ldots, Y_t$ integers denoted by $\mathrm{ML}(Y_i, Y_{i+1})$, $1 \leq i < t$, are maintained satisfying $\mathrm{ML}(Y_i, Y_{i+1}) = \mathrm{ML}(\mathrm{frontier}(Y_i) \cup \mathrm{frontier}(Y_{i+1}))$. For a P-node X a single integer denoted by ML(X) that satisfies $\mathrm{ML}(X) = \mathrm{ML}(\mathrm{frontier}(X))$.

Furthermore, define $\mathrm{LL}(R_i^j)$, the *low indexed level*, to be the smallest d such that R_i^j contains a vertex in V^d and maintain this integer at the root of the corresponding PQ-tree. The *height* of a component R_i^j in the subgraph G^j is $j - \mathrm{LL}(R_i^j)$. The LL-value merely describes the size of the component.

Using these LL- and ML-values, Heath and Pemmaraju [5] have developed operations for merging PQ-trees. These merge operations have been modified and adapted into a larger framework by Jünger et al. [6] to develop an $\mathcal{O}(|V|)$ time algorithm for testing level planarity of not necessarily proper level graphs.

3 Concept of the Algorithm

One can easily obtain the following naive embedding algorithm for level planar graphs. Choose any total order on V^k that is consistent with the set of permutations of V^k that appear in a level planar embedding of $G^k = G$. Choose then any total order on V^{k-1} that is consistent with the set of permutations of V^{k-1} that appear in a level planar embedding of G^{k-1} and that, together with the chosen order of V^k implies a level planar embedding on the subgraph of G

induced by $V^{k-1} \cup V^k$. Extend this construction one level at a time until a level planar embedding of G results.

However, to perform this algorithm, it is necessary to keep track of the set of PQ-trees of every level l, $1 \leq l \leq k$. Besides, an appropriate total order of the vertices of V^j, $1 \leq j < k$, can only be detected by reducing subsets of the leaves of G^j, where the subsets are induced by the adjacency lists of the vertices of V^{j+1}. More precisely, for every pair of consecutive edges $e_1 = (v_1, w)$, $e_2 = (v_2, w)$, $v_1, v_2 \in V^j$, in the adjacency list of a vertex $w \in V^{j+1}$, we have to reduce the set of leaves corresponding to the vertices v_1, v_2 in $\mathcal{T}(G^j)$. This immediately yields an $\Omega(n^2)$ algorithm for nonproper level graphs, with $\Omega(n^2)$ dummy vertices for long edges, since we are forced to consider for every long edge its exact position on the level that is traversed by the long edge.

Instead, we proceed as follows: Let $G = (V, E)$ be a level planar graph with leveling $\mathrm{lev}_G : V \to \{1, 2, \ldots, k\}$. We augment G to a planar directed acyclic st-graph $G_{st} = (V_{st}, E_{st})$ where $V_{st} = V \uplus \{s, t\}$ and $E \subset E_{st}$ such that every source in G has exactly one incoming edge in $E_{st} \setminus E$, every sink in G has exactly one outgoing edge in $E_{st} \setminus E$, $\mathrm{lev}_{G_{st}}(s) = 0$, $\mathrm{lev}_{G_{st}}(t) = k+1$ and for all $v \in V$ we have $\mathrm{lev}_{G_{st}}(v) = \mathrm{lev}_G(v)$. This process, in which two vertices and $\mathcal{O}(|V|)$ edges are added to G, is the nontrivial part of the algorithm that will be explained in section 4.

We compute a topological sorting, i.e., an onto function $\mathrm{ts}_{G_{st}} : V \to \{0, 1, \ldots, |V|+1\}$. The function $\mathrm{ts}_{G_{st}}$ is comparable with $\mathrm{lev}_{G_{st}}$ in the sense that for every $v, w \in V_{st}$ we have $\mathrm{ts}_{G_{st}}(v) \leq \mathrm{ts}_{G_{st}}(w)$ if and only if $\mathrm{lev}_{G_{st}}(v) \leq \mathrm{lev}_{G_{st}}(w)$. Obviously $\mathrm{ts}_{G_{st}}$ is an st-numbering of G_{st}. Using this st-numbering, we can obtain a planar embedding $\mathcal{E}_{st}$ of G_{st} with the edge (s, t) on the boundary of the outer face by applying the algorithm of Chiba et al. [2].

From the planar embedding we obtain a level planar embedding of G_{st} by applying a function "CONSTRUCT-LEVEL-EMBED" that uses a depth first search procedure starting at vertex t and proceeding from every visited vertex w to the unvisited neighbor that appears first in the clockwise ordering of the adjacency list of w in $\mathcal{E}_{st}$. Initially, all levels are empty. When a vertex w is visited, it is appended to the right of the vertices assigned to level $\mathrm{lev}_{G_{st}}(w)$. The restriction of the resulting level orderings to the levels 1 to k yields a level planar embedding of G.

It is clear that the described algorithm runs in $\mathcal{O}(|V|)$ time if the nontrivial part, namely the construction of G_{st} can be achieved in $\mathcal{O}(|V|)$ time. After adding the vertices s and t we augment G to a hierarchy by adding an outgoing edge to every sink of G without destroying level planarity using a function AUGMENT, processing the graph top to bottom. Using the same function AUGMENT again, we process the graph bottom to top and augment G_{st} to an st-graph by adding the edge (s, t) and an incoming edge to every source of G without destroying the level planarity. Thus our level planar embedding algorithm can be sketched as follows.

$\mathcal{E}_l$ **LEVEL-PLANAR-EMBED**$(G = (V^1, V^2, \ldots, V^k; E))$
begin
    ignore all isolated vertices;
    expand G to G_{st} by adding $V^0 = \{s\}$ and $V^{k+1} = \{t\}$;
    AUGMENT(G_{st});
    if AUGMENT fails then
        return $\mathcal{E}_l = \emptyset$;
    $//G_{st}$ is now a hierarchy;
    orient the graph G_{st} from the bottom to the top;
    AUGMENT(G_{st});
    $//G_{st}$ is now an st-graph;
    orient the graph G_{st} from the top to the bottom;
    add edge (s, t);
    compute a topological sorting of V_{st};
    compute a planar embedding $\mathcal{E}_{st}$ according to Chiba et al. [2]
        using the topological sorting as an st-numbering;
    $\mathcal{E}_l$ = CONSTRUCT-LEVEL-EMBED$(\mathcal{E}_{st}, G_{st})$;
    return $\mathcal{E}_l$;
end.

4 Augmentation

In order to add an outgoing edge for every sink of G without destroying level planarity, we need to determine the position of a sink $v \in V^j$, $j \in \{1, 2, \ldots, k-1\}$, in the PQ-trees. This is done by inserting an indicator as a leaf into the PQ-trees. The indicator is ignored throughout the application of the level planarity test and will be removed either with the leaves corresponding to the incoming edges of some vertex $w \in V^l$, $l > j$, or it can be found in the final PQ-tree.

We explain the idea of the approach by an example. Let $v \in V^j$ be a sink. The leaf corresponding to the sink v will be removed from the PQ-tree before testing the graph G^{j+1} for level planarity. Instead of removing the leaf, we keep it in the tree and ignore its presence in the PQ-tree from now on. Such a leaf that marks the position of a sink v in a PQ-tree is called a *sink indicator* and denoted by si(v). The indicator of v may appear within the sequence of leaves corresponding to incoming edges of a vertex $w \in V^l$. The indicator of v is interpreted as a leaf corresponding to an edge $e = (v, w)$ and G is augmented by e. Adding the edge e to G does not destroy the level planarity and provides an outgoing edge for the sink v.

When replacing a leaf corresponding to a sink by a sink indicator, a P- or Q-node X may be constructed in the PQ-tree such that frontier(X) consists only of sink indicators. The presence of such a node in the PQ-tree is ignored as well. A node of a PQ-tree is an *ignored* node if and only if its frontier contains only sink indicators. By definition, a sink indicator is also an ignored node.

In order to achieve linear time for the level planar embedder, we must avoid searching for sink indicators that can be considered for augmentation. Conse-

quently, only the indicators $\mathrm{si}(v)$, $v \in V$, that appear within the pertinent subtree of a PQ-tree with respect to a vertex $w \in V$ are considered for augmentation. It is easy to see that the edges added this way do not destroy level planarity (see Leipert [7]).

However, not all sink indicators are considered for edge insertion. Some of the indicators remain in the final PQ-tree that represents all possible permutations of vertices of V^k in the level planar embeddings of G. It is straightforward to see that if $\mathrm{si}(v)$ is a sink indicator of a vertex $v \in V^j$, $1 < j < k$ and if $\mathrm{si}(v)$ is in the final PQ-tree T, the edge $e = (v, t)$ can be added without destroying level planarity.

While the treatment of sink indicators during the application of the template matching algorithm is rather easy in principle, this does not hold for merge operations. We consider one of the merge operations and discuss necessary adaptations in order to treat the sink indicators correctly. For manipulating sink indicators and ignored nodes correctly during the merge process, ML-values as they have been introduced for nonignored nodes are introduced for ignored nodes as well.

Let R_1^j and R_2^j be two components of G^j that are incident to a vertex w on level $j+1$ and let T_1 and T_2 be their corresponding PQ-trees. Without loss of generality, we may assume that $\mathrm{LL}(T_1) \leq \mathrm{LL}(T_2)$. Thus component R_2^j is the smaller component and an embedding of R_1^j must be found such that R_2^j can be nested within the embedding of R_1^j. This corresponds to adding the root of T_2 as a child to a node of the PQ-tree T_1 constructing a new PQ-tree T'. In order to find an appropriate location to insert T_2 into T_1, we start with the leaf labeled w (that corresponds to the vertex w in R_1^j) in T_1 and proceed upwards in T_1 until a node X' and its parent X are encountered that satisfy certain conditions.

In this extended abstract, we consider only the most difficult condition: Let X be a Q-node with ordered children $X_1, X_2, \ldots, X_\eta$, $X' = X_\lambda$, $1 < \lambda < \eta$, and $\mathrm{ML}(X_{\lambda-1}, X_\lambda) < \mathrm{LL}(T_2)$ and $\mathrm{ML}(X_\lambda, X_{\lambda+1}) < \mathrm{LL}(T_2)$. The node X_λ is replaced by a Q-node Y with two children, X_λ and the root of T_2.

Let $I_1, I_2, \ldots, I_\mu$, $\mu \geq 0$, be the sequence of ignored nodes between $X_{\lambda-1}$ and X_λ in which $X_{\lambda-1}$ and I_1 are direct siblings, and X_λ and I_μ are direct siblings. Let $J_1, J_2, \ldots, J_\rho$, $\rho \geq 0$, be the sequence of ignored nodes between X_λ and $X_{\lambda+1}$ in which X_λ and J_1 are direct siblings, and $X_{\lambda+1}$ and J_ρ are direct siblings.

Let R_{X_i}, $i \in \{1, 2, \ldots, \eta\}$, denote the subgraph of R_1 corresponding to X_i. As illustrated in Fig. 1, there may exist a ν, $1 \leq \nu \leq \mu$, such that for every sink indicator

$$\mathrm{si}(v) \in \bigcup_{i=\nu}^{\mu} \mathrm{frontier}(I_i) \ , \quad v \in \bigcup_{i=1}^{\mathrm{lev}(w)-1} V^i \ ,$$

G has to be augmented by an edge $e = (v, w)$ if R_2 is embedded between $R_{X_{\lambda-1}}$ and R_{X_λ}.

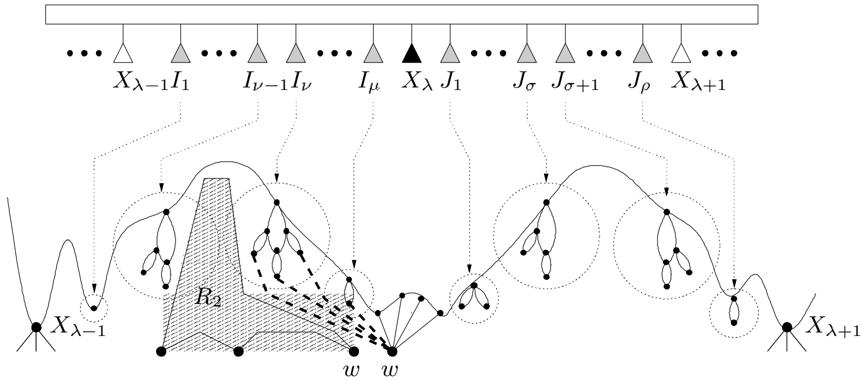


Fig. 1. Merging R_2 into R_1 and embedding it between $R_{X_{\lambda-1}}$ and R_{X_λ} forces G to be augmented by the edges drawn as dotted lines.

Obviously, R_2 can also be embedded between R_{X_λ} and $R_{X_{\lambda+1}}$, and there may exist a σ, $1 \leq \sigma \leq \rho$, such that for every sink indicator

$$\mathrm{si}(v) \in \bigcup_{i=1}^{\sigma} \mathrm{frontier}(J_i) \ , \quad v \in \bigcup_{i=1}^{\mathrm{lev}(w)-1} V^i \ ,$$

G has to be augmented by an edge $e = (v, w)$.

However, it is not possible to decide which set of sink indicators has to be considered for edge augmentation. Proceeding the level planarity test down the levels $V^{\mathrm{lev}(w)+1}$ to V^k may embed the component R_2 on either of the two sides of R_{X_λ}. Since the side is unknown during the merge operation, we have to keep the affected sink indicators in mind. Furthermore, we must devise a method that allows the determination of the correct embedding during subsequent reductions.

The sequences $I_\nu, I_{\nu+1}, \ldots, I_\mu$ and $J_1, J_2, \ldots, J_\sigma$ are called the *reference sequences* of R_2 and denoted by $\mathrm{rseq}(R_2)$. We refer to $I_\nu, I_{\nu+1}, \ldots, I_\mu$ as the *left reference sequence* of R_2 denoted by $\mathrm{rseq}(R_2)^{left}$, and to $J_1, J_2, \ldots, J_\sigma$ as the *right reference sequence* denoted by $\mathrm{rseq}(R_2)^{right}$. The union $\bigcup_{i=\nu}^{\mu} \mathrm{frontier}(I_i) \cup \bigcup_{i=1}^{\sigma} \mathrm{frontier}(J_i)$ is called the *reference set* of R_2 and denoted by $\mathrm{ref}(R_2)$. The *left* and *right reference set* $\mathrm{ref}(R_2)^{left}$ and $\mathrm{ref}(R_2)^{right}$, respectively, are defined analogously to the left and right reference sequence.

In order to solve the decision problem in the described merge operation, we examine how R_2 is fixed to either side of the vertex $w \in V$ in a level planar embedding of G. The following two nontrivial lemmas (see [7]) are based on two template replacement patterns Q2 and Q3 as they are used in the template reduction of Boot and Lueker [1].

Lemma 1. *The subgraph R_2 must be fixed in its embedding at one side of R_{X_λ} with respect to R_X if and only if the Q-node Y is removed from the tree T during the application of the template matching algorithm using template Q2 or template*

Q3, and the parent of Y did not become a node with Y as the only nonignored child.

Lemma 2. *The subgraph R_2 is not fixed to any side of R_{X_λ} with respect to R_X if and only if one of the following conditions applies during the application of the template matching algorithm.*

- *The Q-node Y gets ignored.*
- *The Q-node Y is a nonignored node of the final PQ-tree.*
- *The Q-node Y has only one nonignored child.*
- *The parent of Y has only Y as a nonignored child.*

Lemmas 1 and 2 reveal a solution for solving the problem of deciding whether or not R_2 is fixed to one side of R_{X_λ} with respect to R_X. A strategy is developed for detecting on which side of R_{X_λ} the subgraph R_2 has to be embedded. One endmost child of Y can clearly be identified with the side where the root of T_2 has been placed, while the other endmost child of Y can be identified with the side were X_λ is. Every reversion of the Q-node Y corresponds to changing the side were R_2 must be embedded and all we need to do is to detect the side of Y that belongs to R_2, when finally removing Y from the tree by applying one of the templates Q2 or Q3. The strategy is to mark the end of Y belonging to R_2 with a special ignored node. Such a special ignored node is called a *contact* of R_2 and denoted by $\mathrm{c}(R_2)$. During the merge operation, it is placed as an endmost child of Y next to the root of T_2. Thus the Q-node Y has now three children instead of two. See Fig. 2 for an illustration.

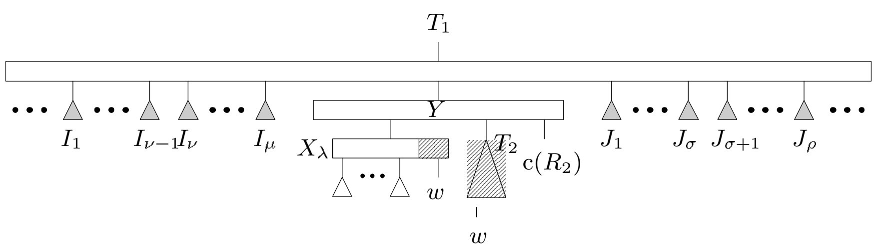


Fig. 2. Contact $\mathrm{c}(R_2)$ is added as a child to Y next to the root of T_2 after the PQ-trees T_1 and T_2 have been merged.

Before gathering some observations about contacts, it is necessary to show that the involved ignored nodes remain in the relative position of Y within the Q-node, and are therefore not moved or removed.

Lemma 3. *The ignored nodes of* $\mathrm{rseq}(R_2)^{left}$ *and* $\mathrm{rseq}(R_2)^{right}$ *stay siblings of Y until one of the templates Q2 or Q3 is applied to Y and its parent.*

A contact has some special attributes that are immediately clear and very useful for our approach. In the following observations we again assume that Y and its parent have not been subject to another merge operation.

Observation 1 *Since the contact is an endmost child of a Q-node Y, it will remain an endmost child of the same Q-node Y, unless the node Y is eliminated by applying one of the templates Q2 or Q3.*

Observation 2 *If the node Y is eliminated by applying one of the templates Q2 or Q3, the contact* $\mathrm{c}(R_2)$ *determines the side were R_2 must be embedded next to R_{X_λ} with respect to R_X. The contact is then a direct sibling to* $\mathrm{rseq}(R_2)^i$ *for some $i \in \{left, right\}$ and* $\mathrm{ref}(R_2)^i$ *must be considered for edge augmentation.*

Observation 3 *If one of the four cases mentioned in Lemma 2 applies to Y or its parent, R_2 can be embedded on any side R_{X_λ} with respect to R_X and therefore either* $\mathrm{ref}(R_2)^{left}$ *or* $\mathrm{ref}(R_2)^{right}$ *must be considered for edge augmentation.*

Besides placing $\mathrm{c}(R_2)$ as an endmost child next to the root of T_2, $\mathrm{c}(R_2)$ is equipped with a set of four pointers, denoting the beginning and the end of both the left and the right reference sequence of R_2. After performing a reduction by applying one of the templates Q2 or Q3 to the node Y, the contact is either a direct sibling of I_μ or a direct sibling of J_1. In the first case, we scan the sequence of ignored siblings starting at I_μ until the ignored node I_ν is detected. In the latter case, the sequence of ignored siblings is scanned by starting at J_1 until the node J_σ is detected. Figure 3 illustrates this strategy for the latter case. By storing pointers of the ignored nodes $I_\nu, I_\mu, J_1, J_\sigma$ at $\mathrm{c}(R_2)$, we are able to identify the reference set $\mathrm{ref}(R_2)$.

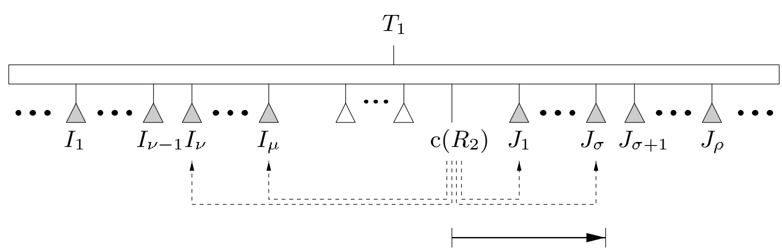


Fig. 3. Identification of the reference set that has to be chosen for augmentation. Contact $\mathrm{c}(R_2)$ is adjacent to the ignored node J_1 after the application of template Q2 or Q3. We chose $\mathrm{ref}(R_2)^{right}$ for augmentation. The dotted lines denote the pointers of $\mathrm{c}(R_2)$ to $I_\nu, I_\mu, J_1, J_\sigma$.

For clarity, we omitted the concatenation of merge operations. However, straightforward methods in the application of contacts are able to handle such concatenations (see [7] for details).

Theorem 4. *The algorithm LEVEL-PLANAR-EMBED computes a level planar embedding of a level planar graph $G = (V, E)$ in $\mathcal{O}(|V|)$ time.*

Proof. The correctness follows from the correctness of the function AUGMENT and the discussion of section 3. The linear running time of AUGMENT follows from an amortized analysis based on the linear running time of the level planar testing algorithm.

5 Remarks

Once a level graph has been level planar embedded, we want to visualize it by producing a level planar drawing. There is a nice and quick solution to this problem that uses some extra information that is computed by our level planar embedding algorithm. Instead of drawing the graph G directly, we draw the st-graph G_{st} using the planar embedding of G_{st}, and remove the vertices s and t as well as all edges in $E_{st} \setminus E$ afterwards.

Suitable approaches for drawing an st-graph G_{st} have been presented in [3] and [4]. These algorithms construct a planar upward polyline drawing of a planar st-graph according to a topological numbering of the vertices. The vertices of the st-graph are assigned to grid coordinates and the edges are drawn as polygonal chains. If we assign a topological numbering to the vertices according to their leveling, the algorithm presented by Di Battista and Tamassia [3] produces in $\mathcal{O}(|V|)$ time a level planar polyline grid drawing of G_{st} such that the number of edge bends is at most $6n - 12$ and every edge has at most two bends. This approach can be improved to produce in $\mathcal{O}(|V|)$ time a level planar polyline grid drawing of G_{st} such that the drawing of G_{st} has $\mathcal{O}(|V|^2)$ area, the number of edge bends is at most $(10n-31)/3$, and every edge has at most two bends. Thus once we have augmented G to the st-graph G_{st}, we can immediately produce a level planar drawing of G in $\mathcal{O}(|V|)$ time.

References

1. K. Booth and G. Lueker. Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms. *Journal of Computer and System Sciences*, 13:335–379, 1976.
2. N. Chiba, T. Nishizeki, S. Abe, and T. Ozawa. A linear algorithm for embedding planar graphs using PQ-trees. *Journal of Computer and System Sciences*, 30:54–76, 1985.
3. G. Di Battista and R. Tamassia. Algorithms for plane representations of acyclic digraphs. *Theoretical Computer Science*, 61:175–198, 1988.
4. G. Di Battista, R. Tamassia, and I. G. Tollis. Constrained visibility representations of graphs. *Information Processing Letters*, 41:1–7, 1992.
5. L. S. Heath and S. V. Pemmaraju. Recognizing leveled-planar dags in linear time. In F. J. Brandenburg, editor, *Proc. Graph Drawing '95*, volume 1027 of *Lecture Notes in Computer Science*, pages 300–311. Springer Verlag, 1995.
6. M. Jünger, S. Leipert, and P. Mutzel. Level planarity testing in linear time. In S. Whitesides, editor, *Graph Drawing '98*, volume 1547 of *Lecture Notes in Computer Science*, pages 224–237. Springer Verlag, 1998.
7. S. Leipert. *Level Planarity Testing and Embedding in Linear Time.* PhD thesis, Universität zu Köln, 1998.

Higres – Visualization System for Clustered Graphs and Graph Algorithms*

Ivan A. Lisitsyn and Victor N. Kasyanov

A. P. Ershov's Institute of Informatics Systems,
Lavrentiev av. 6, 630090, Novosibirsk, Russia
lis@iis.nsk.su

Abstract. We present the Higres system – a visualization tool, an editor for clustered graphs and a platform for execution and animation of graph algorithms. Higres can handle any attributed graphs and perform animated semantic processing of them. The semantics of a graph can be defined by the user in the graph editor. The user can also create new external modules to process graphs with defined semantics. In addition we provide a possibility to extend system with new graph drawing algorithms by using special type of external modules for this purpose. We provide an easy to use C++ library for implementation of both types of extensions.

1 Introduction

Graph models can be used in practice only along with support tools that provide visualization, editing and processing of graphs. For this reason many graph visualization systems, graph editors and libraries of graph algorithms have been developed in recent years. Examples of these tools include VCG [1], daVinci [2], Graphlet [3], GraVis [4], GLT & GET [5] and LEDA [6].

In some application areas the information is organized too complex to be modeled by a classical graph. In this case other graph-like structures should be taken in consideration. Hence, there is a need of tools that are capable of visualization of such structures. For example, GLT & GET can visualize graphs, vertices of which are associated with other graphs; D-ABDUCTOR [7] is a system designed for visualization of compound graphs.

One of recent graph formalisms is the *clustered graphs* . A clustered graph consists of a classical graph and a recursive clustering structure that partitioning vertices of this graph into hierarchically enclosed fragments. Clustered graphs are used in many areas where strong structuring of information is needed. Some theoretical results on clustered graphs with application to graph drawing are presented in [8,9,10,11]. A system that provides some kind of animated visualization and force-based allocation of clustered graphs is presented in [13].

Several general purpose graph visualization systems provide recursive folding of subgraphs allowing to create clusters. However, this feature is commonly used

* Supported by RFBR Grant N 98-01-748

J. Kratochvíl (Ed.): GD'99, LNCS 1731, pp. 82–89, 1999.

only to hide a part of information and not always applicable to the visualization of clustered graphs.

Usual graph editors either do not have support for attributed graphs or have rather weak support. Though the GML file format used by Graphlet can store arbitrary number of labels associated with each graph element, it is impossible to edit and visualize these labels in Graphlet's graph editor. Most editors allow only one text label for each vertex and optionally for each edge.

In this paper we present the Higres system, which is a visualization tool, an editor for attributed clustered graphs and a platform for execution and animation of graph algorithms. Proper support for attributed graphs allows us to define the semantics of a graph inside the graph editor. This feature can be particularly useful in conjunction with animation of graph algorithms, since in this case we can animate any kind of semantic processing.

Algorithms executed by Higres are implemented in external modules. The user can create his/her own modules with algorithms that process graphs with user defined semantics. In addition we provide special type of integration for graph drawing algorithms. Hence, the system can be extended with two types of modules. First type provides animation capabilities whereas second type is easier to use (user only needs to open a module and wait for the result). A new graph drawing method can be developed and tested with help of a module of the first type, then used in practice as a "quick" module without animation. We provide a special C++ library for implementation of both types of extension modules.

Higres is implemented in C++ with MSVC compiler and MFC library. The system works under MS Windows 95/98 and Windows NT.

2 Clustered Graphs in Higres

A clustered graph consists of vertices, fragments and edges, which we call objects. Vertices and edges form a classical graph that can be either directed or undirected. We allow multiple edges and loops. Each fragment is associated with a set of vertices. We say that these vertices belong to the fragment. If the set of vertices belonging to a fragment intersects with the set of vertices of another fragment, then either first fragment is a subfragment of second one or vise versa. The set of all vertices forms the *main fragment* of a clustered graph.

The semantics of a clustered graph is represented in Higres by means of object types. Each object in a graph belongs to an object type. A set of labels is defined for each object type. Each label has its data type, name and several other parameters. A set of values is associated with each object according to the set of labels defined for the object type to which this object belongs. These values along with partitioning of objects into types represent the semantics of a graph. Thus, user can define the semantics by creating new object types and adding new labels.

3 Visualization

In Higres each fragment is represented by a rectangle. All vertices that belong to a fragment and all its subfragments are located inside its rectangle. Fragments as well as vertices never overlap each other. Each fragment can be either closed or open. When a fragment is open, its content is visible to the user; when it is closed, it is drawn as an empty rectangle with only labels text inside it. A separate window can be opened to observe each fragment. Only the content of a fragment is shown inside its window, though it is possible to see this content inside windows of parent fragments, if the fragment is open. Fig. 1 shows a clustered graph with open and closed fragments.

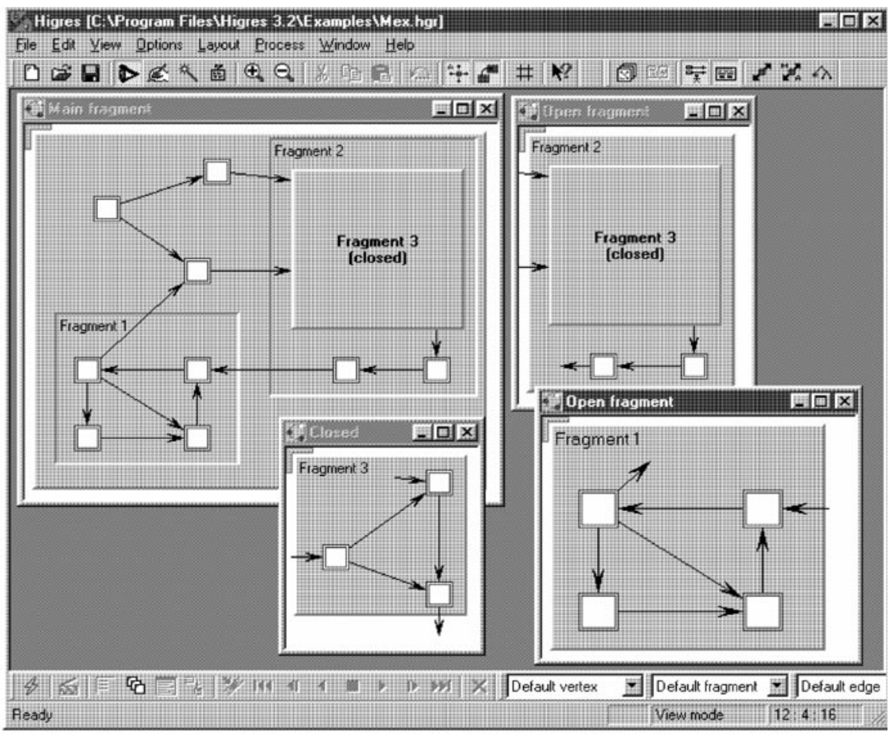


Fig. 1. Higres working with a clustered graph.

Most part of visual attributes of an object is defined by its type. This means that semantically related objects have similar visual representation. Higres uses flexible technique to visualize object labels. The user specifies a text template for each object type. This template is used to create labels text of objects of the given type by inserting in it labels values of a particular object.

Other visualization features include the following:
- various shapes and styles for vertices;
- polyline and smooth curved edges;
- various styles for edge lines and arrows;
- color selection for all graph components;
- possibility to scale graph image to arbitrary size;
- edge text movable along the edge line;
- external vertex text movable around the vertex;
- font selection for labels text;
- two graphical output formats;
- a number of options that control graph visualization.

Currently Higres uses three graph drawing algorithms for automatic graph allocation. First one is a force method, which is very close to original algorithm from [12]. Second one is our improvement of the first. Third method allocates rooted trees on layers. These algorithms are implemented in external modules.

4 User Interface

A comfortable and intuitive user interface was one of our main objectives in developing Higres.

The system's main window contains a several toolbars that provide quick access to frequently used menu commands, and object types switching for creation of new objects. The status bar displays menu and toolbars hints along with other useful information on current editor operation and graph processing status.

The system uses two basic modes: view and edit. In the view mode it is only possible to open/close fragments and fragment windows, but the scrolling operations are extended with mouse scrolling.

In the edit mode the left mouse button is used to select objects and the right mouse button displays the popup menu, in which the user can choose the operation he/she wants to perform with the selected objects or with whole graph. It is also possible to create new objects by selecting commands from that menu (fig.2).

The left mouse button can be also used to move vertices, fragments, labels texts and edge bends, and resize vertices and fragments. Thus, all edit operations are gathered in a single mode. To our opinion it is more useful approach (especially for inexperienced users), than division to several modes. However, for adherents of the latter we provide two additional modes. Their usage is optional but in some cases they may be useful: creation mode for object creation and labels mode for labels editing.

Other interface features include the following:
- cut/copy/paste operations available for any graph parts;
- almost unlimited number of undo levels;
- optimized screen update;
- automatic elimination of objects overlapping;
- automatic vertex size adjusting;

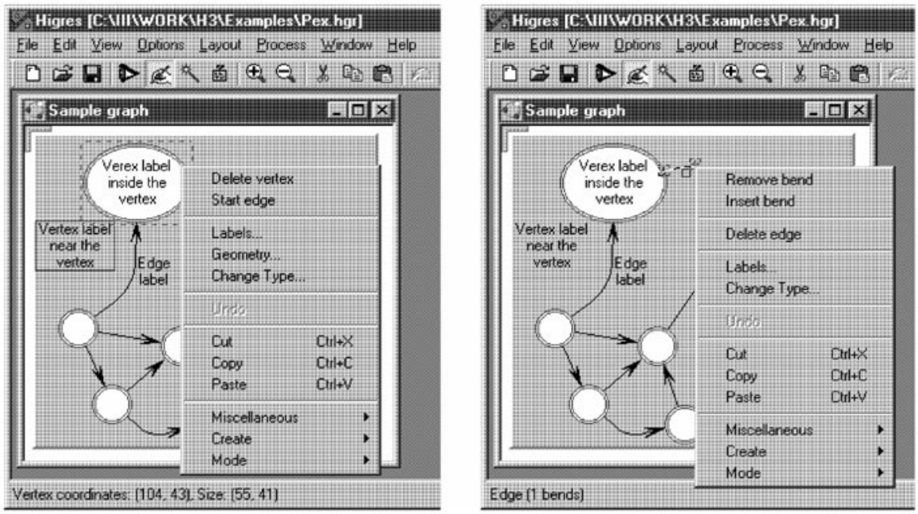


Fig. 2. Popup menus in the edit mode: a). menu for a vertex; b). menu for an edge.

- grid with several parameters;
- a number of options that configure user interface;
- online help available for each menu, dialog box and editor mode.

5 Algorithms Animation

To run an algorithm the user should select an external module in the dialog box. The system starts this module and optionally opens the process protocol window for textual process output. Higres provides both run-time animation of algorithms and sample caching for repeated and backward animation.

A set of parameters can be defined inside a module. These parameters can be changed by the user at any execution step. The module can input text and numbers from the user. It can also write any textual information to the protocol window.

A wide range of semantic and graph drawing algorithms can be implemented as external modules. Screen shots on fig. 3,4,5 give some examples of semantic algorithms executed on different graph models.

The animation feature can be used for algorithm testing and debugging, educational purposes and exploration of iteration processes such as force methods in graph drawing.

We provide a special C++ library that can be used to create external modules. This library includes functions for graph modification and ones that provide interaction with the system. It is unnecessary for programmer, who use this library, to know the details of the internal representation of graphs in Higres. This approach greatly simplifies the creation of new modules.

6 Conclusion

We presented the Higres system, which is a visualization tool, an editor for attributed clustered graphs and a platform for execution and animation of graph algorithms. The system is fully usable and has applications to education, research and practice. Our future plans in improving and extending the system concern the integration of additional graph drawing algorithms, including methods specially designed for allocation of clustered graphs.

The system is available on WWW at http://lis.iis.nsk.su/higres.

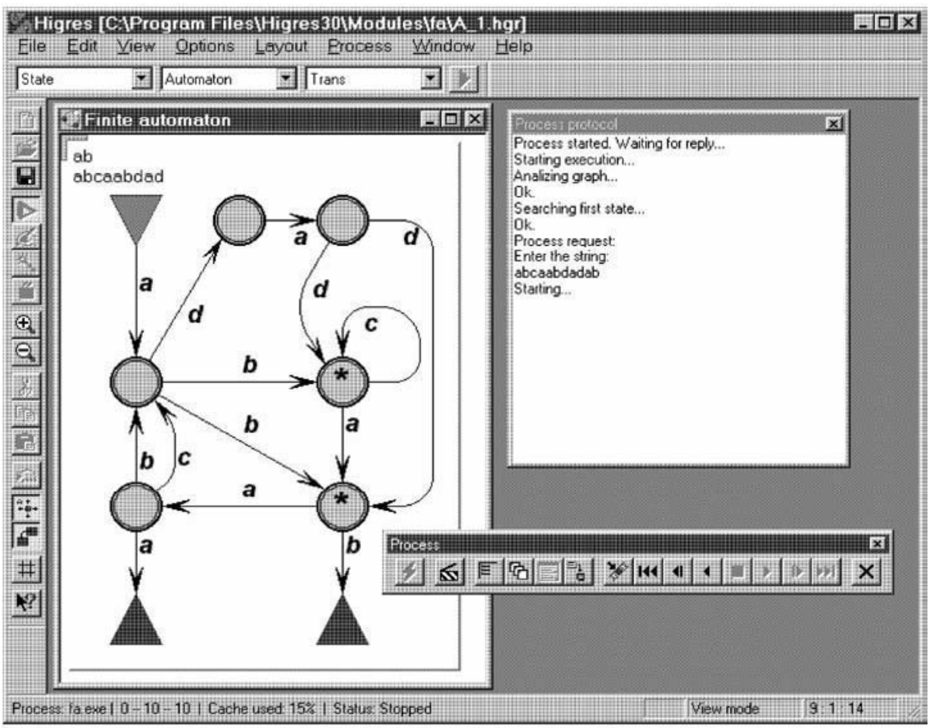


Fig. 3. The graph of a finite automaton. The external module emulates the work of this automaton. Asterisks show active states. Two lines in the top left corner are the tail of the string entered by the user and the part of that string that is already processed.

References

1. G. Sander. Graph layout through the VCG tool. Proc. of Graph Drawing '94, volume 894 of LNCS, pages 194-205, 1995.

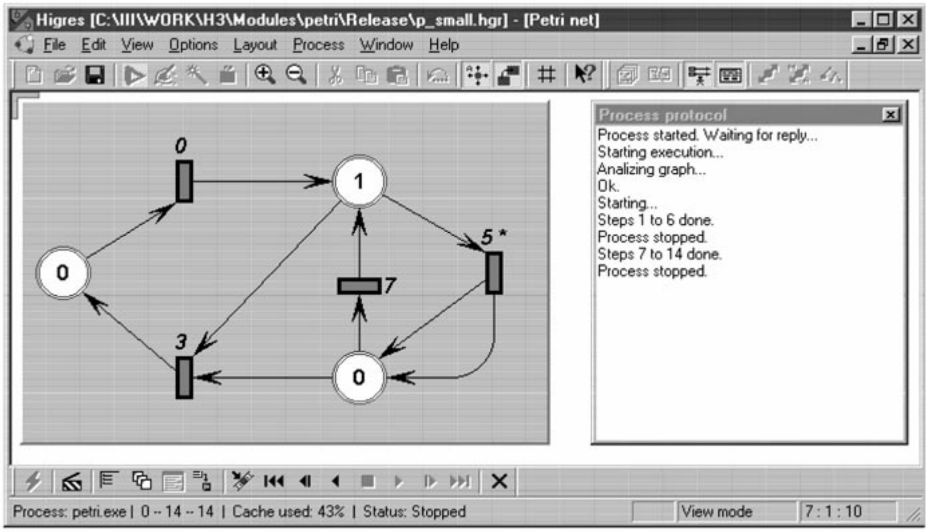


Fig. 4. A Petri net. The external module performs its simulation. Asterisk shows the transition that will fire on the next step. Numbers inside places are numbers of tokens. Numbers near transitions are their priorities introduced in order to make this Petri net determinate.

2. M. Fröhlich and M. Werner. Demonstration of the interactive graph visualization system daVinci. Proc. of Graph Drawing '94, volume 894 of LNCS, pages 266-269, 1995.
3. M. Himsolt. The Graphlet System (system demonstration). Proc. of Graph Drawing '96, volume 1190 of LNCS, pages 233-240, 1997.
4. H. Lauer, M. Ettrich and K. Soukup. GraVis – System Demonstration. Proc. of Graph Drawing '97, volume 1353 of LNCS, pages 344-349, 1997.
5. B. Madden, P. Madden, S. Powers and M. Himsolt. Portable Graph Layout and Editing. Proc. of Graph Drawing '95, volume 1027 of LNCS, pages 385-395, 1996.
6. K. Mehlhorn and S. Näher. LEDA: a platform for combinatorial and geometric computing. Commun. ACM, 38:96-102, 1995.
7. K. Siguyama and K. Misue. A Generic Compound Graph Visualizer/Manipulator: D-ABDUCTOR. Proc. of Graph Drawing 95, volume 1027 of LNCS, pages 500-503, 1996.
8. P. Eades, Q.W. Feng. Drawing Clustered Graphs on an Orthogonal Grid. Proc. of Graph Drawing 97, volume 1353 of LNCS, pages 182-193, 1997.
9. P. Eades, Q.W. Feng. Multilevel visualization of Clustered Graphs. Proc. of Graph Drawing 96, volume 1190 of LNCS, pages 101-112, 1997.
10. P. Eades, Q.W. Feng and X. Lin. Straight-Line Drawing Algorithms for Hierarchical Graphs and Clustered Graphs. Proc. of Graph Drawing 96, volume 1190 of LNCS, pages 113-128, 1997.
11. Q.W. Feng, R. Cohen and P. Eades. How to draw a planar clustered graph. In COCOON'95, volume 959 of LNCS, pages 21-31, 1995.
12. P. Eades. A Heuristic For Graph Drawing. Congressus Numerantium, 42, pages 149-160, 1984.

13. M.L. Huang, P. Eades. A Fully Animated Interactive System for Clustering and Navigating Huge Graphs. Proc. of Graph Drawing 98, volume 1547 of LNCS, pages 374-383, 1998.

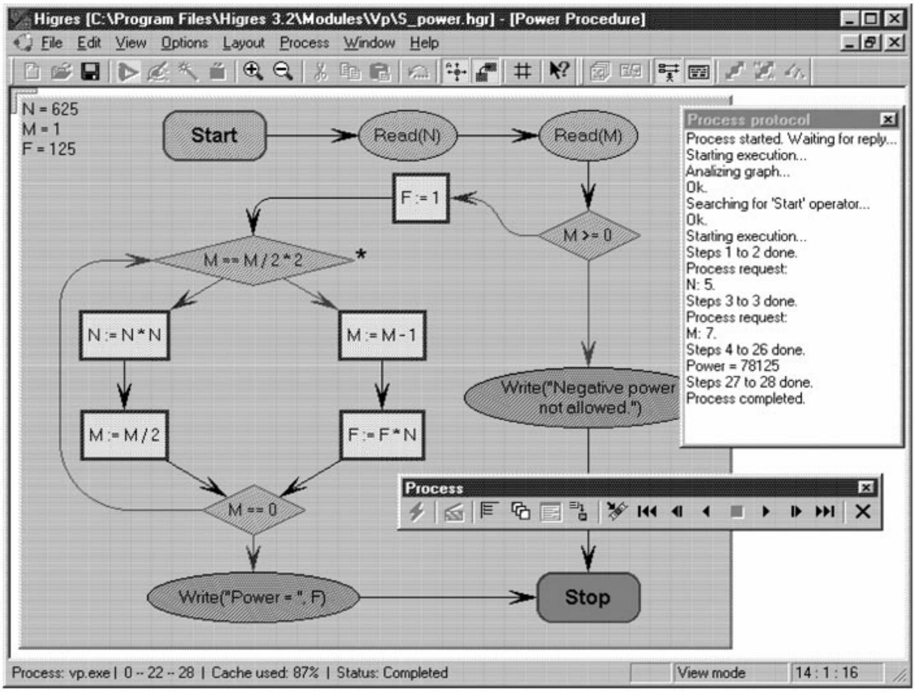


Fig. 5. Program scheme interpreted by external module. Asterisk shows which operator has control. Current values of variables are listed in the top left corner of the fragment rectangle. This screen shot was made after the completion of the process and rewinding several samples back.

Partitioning Approach to Visualization of Large Graphs

Vladimir Batagelj[1], Andrej Mrvar[2], and Matjaž Zaveršnik[1]

[1] FMF, Department of Mathematics, and IMFM,
Department of Theoretical Computer Science
University of Ljubljana
Jadranska 19, 1000 Ljubljana, Slovenia
vladimir.batagelj@uni-lj.si,
matjaz.zaversnik@fmf.uni-lj.si
[2] Faculty of Social Sciences
University of Ljubljana
Kardeljeva ploščad 5, 1000 Ljubljana, Slovenia
andrej.mrvar@uni-lj.si

Abstract. The structure of large graphs can be revealed by partitioning graphs to smaller parts, which are easier to handle. In the paper we propose the use of core decomposition as an efficient approach for partitioning large graphs. On the selected subgraphs, computationally more intensive, clustering and blockmodeling can be used to analyze their internal structure. The approach is illustrated by an analysis of Snyder & Kick's world trade graph.

1 Approaches to Clustering in Graphs

In the analysis of a large graph we often decompose / reduce it to several smaller manageable parts. This can be done even hierarchically. In the paper we discuss subgraphs induced by classes (clusters) of some partition of the graph vertex set.

There exist several approaches for partitioning graphs. They can be divided in the following groups:

- connectivity based partitions:
  - standard concepts from Graph Theory: components, cliques, k-cores, distance partition from selected subset, ...
  - neighborhoods of "central" vertices
- neighborhood based partitions: cluster is a set of units with *similar* neighborhoods (degree partition, regular partition, colorings, ...)
- other approaches:
  - eigen-vector methods
  - hierarchy of similar graphs

Only approaches with subquadratic time complexities (such as $\mathcal{O}(n)$, $\mathcal{O}(n \log n)$ and $\mathcal{O}(n\sqrt{n})$, n is the number of vertices), are fast enough for large graphs.

J. Kratochvíl (Ed.): GD'99, LNCS 1731, pp. 90–97, 1999.

2 Cores

In this paper we propose the use of cores as an efficient approach to decomposition of large graphs. The notion of core was introduced by Seidman in 1983 [8].

Let $G = (V, L)$ be a graph. V is the set of *vertices* and $L = E \cup A$ is the set of *lines* (*edges* or *arcs*). A subgraph $H = (W, L|W)$ induced by the set W is a *k-core* or a *core of order k* iff $\forall v \in W : \deg_H(v) \geq k$ and H is a maximum subgraph with this property. The core of maximum order is also called the *main* core.

The degree $\deg(v)$ can be: in-degree, out-degree, in-degree + out-degree, min(in-degree, out-degree), ... determining different types of cores. The notion of cores can be generalized to take values of lines into account. Similarly the *p-core* can be defined, where $p \in (0, 1)$, with the requirement that every vertex in p-core has the proportion p of all neighbors in the core.

The cores have the following properties:

- The cores are nested: $i < j \implies H_j \subseteq H_i$ (see Figure 1).
- There exists an efficient algorithm to determine the core hierarchy.
- Cores are not necessarily connected subgraphs.

An algorithm for determining the cores hierarchy is based on the following property:

> If from a given graph $G = (V, L)$ we recursively delete all vertices, and lines incident with them, of degree less than k the remaining graph is the k-core.

INPUT: graph $G = (V, L)$ represented by lists of neighbors
OUTPUT: table *core* with core number for each vertex

1.1 compute the degrees of vertices;
1.2 order the set of vertices V in increasing order of their degrees;
2 **for each** $v \in V$ in the order **do begin**
2.1 $\quad$ $core[v] := degree[v]$;
2.2 $\quad$ **for each** $u \in Neighbors(v)$ **do**
2.2.1 $\quad\quad$ **if** $degree[u] > degree[v]$ **then begin**
2.2.1.1 $\quad\quad\quad$ $degree[u] := degree[u] - 1$;
2.2.1.2 $\quad\quad\quad$ update the ordering of V
$\quad\quad$ **end**
$\quad$ **end**;

Let us denote $n = |V|$ and $m = |L|$. We shall show that the described algorithm can be implemented to run in time $O(m)$.

To (1.1) compute the degrees of all vertices we need time $O(m)$. Using *bin sort* – collecting all vertices of the same degree in a separate list, combined into the ordered list by the table of their starts, we can (1.2) order the set V in time $O(n)$. The statement (2.1) requires a constant time and therefore contributes $O(n)$ to the algorithm.

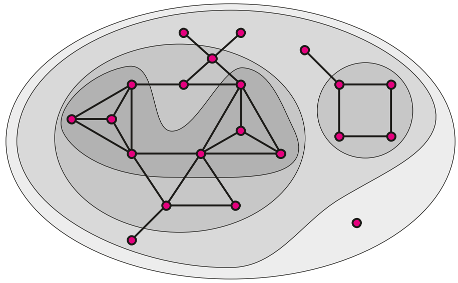


Fig. 1. 0, 1, 2 and 3 core.

The conditional statement (2.2.1) can be implemented to run in constant time by the use of the table determining the position of each vertex data in the order list. Since it is executed for each line at most twice the contribution of (2.2) in all repetitions of (2) is $O(m)$.

Therefore the total time complexity of the algorithm is $O(m)$.

2.1 Core Decomposition

We propose the following approach to decomposition of large graphs based on k-cores: The main core is the most interesting, so we analyze it separately. After that we determine the residual graph, which can be obtained in different ways: by shrinking the main core; by deleting lines in the main core; or by deleting the main core completely (deleting vertices and lines). Again we analyze the main core of the residual graph, and so on ... . Note, as a 'main' core we can consider also the union of some top-level cores selected on the basis of 'core spectrum'.

3 Clusterings in Graphs

Let E be a finite *set of units*. Its nonempty subset $C \subseteq E$ is called a *cluster*. A set of clusters $\mathbf{C} = \{C_i\}$ forms a *clustering*. The clustering $\mathbf{C}$ is a *complete* clustering if it is a partition of the set of units E.

The *clustering problem* $(\Phi, P, \min)$ can be expressed as: Determine the clustering $\mathbf{C}^* \in \Phi$, for which

$$P(\mathbf{C}^*) = \min_{\mathbf{C} \in \Phi} P(\mathbf{C})$$

where Φ is a *set of feasible clusterings* and $P : \Phi \to \mathbb{R}_0^+$ is a *clustering criterion function*. We denote the *set of minimal solutions* by $\mathrm{Min}(\Phi, P)$.

Let $(\mathbb{R}_0^+, \boxplus, 0, \leq)$ be an ordered abelian monoid – usually $\boxplus$ stands for $+$ or max. A *simple* criterion function P has the form:

$$P(\mathbf{C}) = \bigboxplus_{C \in \mathbf{C}} p(C), \quad p(C) \geq 0 \qquad \text{and} \qquad \forall X \in E : p(\{X\}) = 0$$

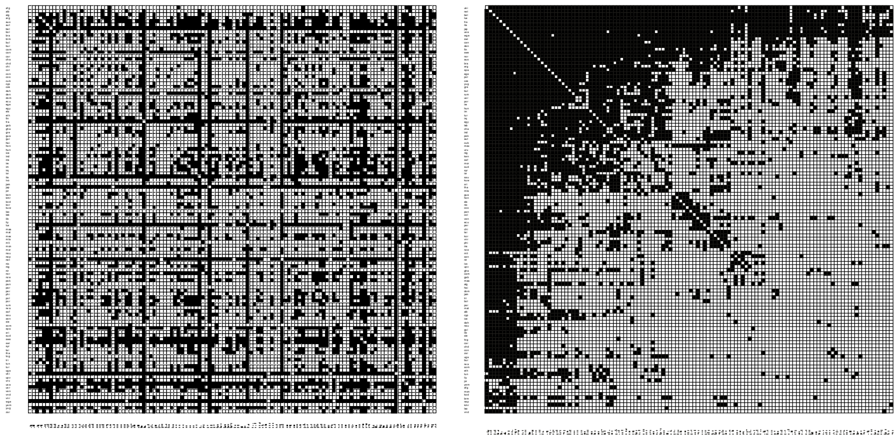


Fig. 2. World trade graph – original adjacency matrix and adjacency matrix reordered according to core decomposition.

The 'cluster error' function $p(C)$ is usually expressed using some dissimilarity measure d over E. For example:

$$p(C) = \max_{u,v \in C} d(u,v) \qquad \text{or} \qquad p(C) = \frac{1}{|C|} \sum_{u,v \in C} d(u,v)$$

An example of dissimilarity between vertices ($\oplus$ denotes the symmetric difference) is:

$$d(u,v) = \frac{|N^+(u) \oplus N^+(v)|}{|N^+(u) \cup N^+(v)|}$$

where $N^+(v) = \{u \in V : (v : u) \in L\} \cup \{v\}$ is the *rooted neighborhood* of vertex $v \in V$.

Assume $E = V$. A pair of clusters (C_1, C_2) determines a *block* – a subgraph

$$B(C_1, C_2) = (C_1 \cup C_2, \{(u,v) \in L : u \in C_1, v \in C_2\})$$

Given a clustering $\mathbf{C}$ we obtain a reduced graph or *blockmodel* of G by shrinking each cluster to a vertex and deciding for each induced block whether it produces a line in the reduced graph, and of what type.

To evaluate the quality of blockmodel a criterion function of *general* form is needed:

$$P(\mathbf{C}; \mathcal{T}) = \sum_{(C_1,C_2) \in \mathbf{C} \times \mathbf{C}} \min_{T \in \mathcal{T}} w(T)\delta(C_1, C_2; T)$$

where $\mathcal{T}$ is a set of feasible types, and δ measures the deviation of blocks, induced by a clustering, from the ideal block structure. For details see [1,3].

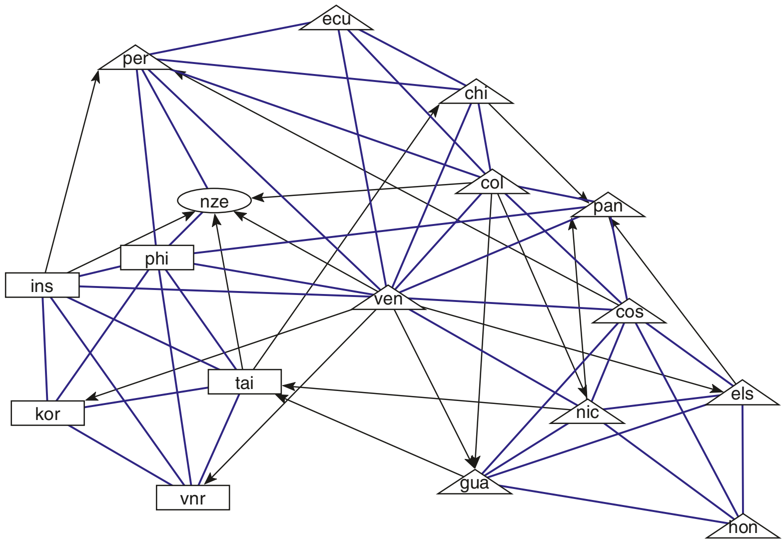


Fig. 3. Secondary core.

4 Example – World Trade Graph

We selected *Snyder & Kick's world trade graph* [9] to illustrate our partitioning approach. World trade among 118 countries is described by 2116 edges and 514 arcs.

Because of limited space for this paper or because of their format some picture were omitted. They are available at:

`http://vlado.fmf.uni-lj.si/pub/networks/doc/part/gd99.htm`

First we tried to obtain a layout of World trade graph using standard spring embedders. The resulting layouts were not satisfactory – the inner structure of the graph cannot be noticed in them.

There are also too many lines. For such graphs a presentation by adjacency matrix is an option. The adjacency matrix of the World trade graph is displayed on the left side of Figure 2. To see some structure in it we have to reorder it.

The first approach to reorder the vertices was by the core decomposition. The main core contains 54 countries (from 118). Its layout was obtained using a 3D spring embedder. The lines were omitted in the picture. By examining countries we found out that such layout could be expected – similar countries were put close to each other.

Because the main core is quite large we divided it further into three clusters using pre-specified blockmodeling [3] with the center-periphery pattern as a goal. The first of the three obtained clusters is a clique on 8 vertices formed by

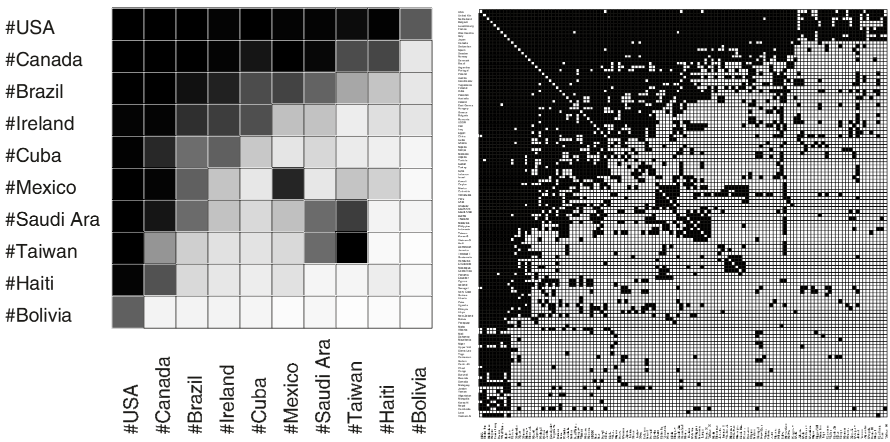


Fig. 4. Blockmodel and reordered adjacency matrix.

Luxembourg, France, Belgium, Denmark, The Netherlands, Great Britain, Italy and Japan. The other two clusters are far away from cliques.

Vertices and lines of the main core were removed from the graph and the secondary core was computed afterwards. It is shown in Figure 3. Three additional steps were needed to complete the decomposition.

The right side of Figure 2 shows the reordered matrix according to the cores – 1's are mostly in the upper left corner. The partition into clusters can be seen in the matrix as well.

Next we applied a blockmodeling (structural equivalence, 10 clusters) and obtained the following clustering $\mathbf{C}_{10}$:

1. USA, UK, Netherlands, Belgium, Luxembourg, France, West Germany, Italy, Japan.
2. Canada, Switzerland, Spain, Sweden, Norway, Denmark.
3. Brazil, Argentina, Portugal, Poland, Austria, Czechoslovakia, Yugoslavia, Finland, India, Pakistan, Australia.
4. Ireland, East Germany, Hungary, Greece, Bulgaria, Rumania, USSR, Iran, Iraq, Egypt, China.
5. Cuba, Ghana, Nigeria, Kenya, Morocco, Algeria, Tunisia, Sudan, Turkey, Syria, Lebanon, Israel, Kuwait, Ceylon.
6. Mexico, Colombia, Venezuela, Peru, Chile, Uruguay, South Africa.
7. Saudi Arabia, Burma, Thailand, Malaysia, Philippines, Indonesia.
8. Taiwan, Korea-S, Vietnam-S.
9. Haiti, Dominican Rep., Jamaica, Trinidad-Tobago, Guatemala, Honduras, El Salvador, Nicaragua, Costa Rica, Panama, Ecuador, Cyprus, Iceland, Senegal, Ivory Coast, Guinea, Liberia, Zaire, Uganda, Ethiopia, Libya, New Zealand.
10. Bolivia, Paraguay, Malta, Albania, Mali, Dahomey, Mauritania, Niger, Upper Volta, Sierra Leone, Togo, Cameroun, Gabon, Centr. Afr. Rep., Chad, Congo, Burundi, Rwanda, Somalia, Malagasy, Jordan, Yemen, Afghanistan, Mongolia, Korea-N, Nepal, Cambodia, Laos, Vietnam-N.

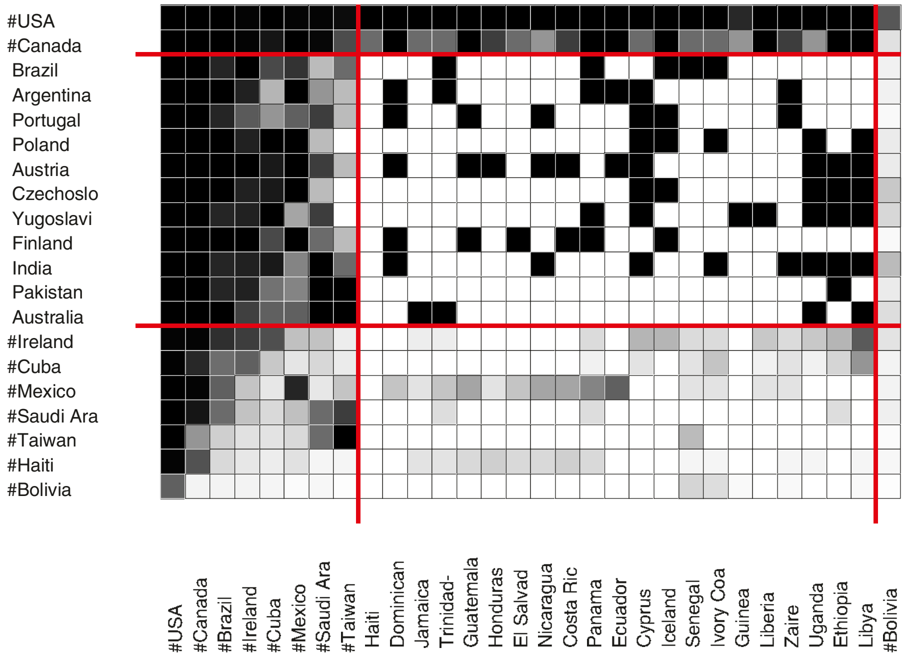


Fig. 5. Contextual matrix (C_3, C_9).

with $P(\mathbf{C}_{10}) = 1452$ – the CONCOR clustering reported in [9] has $P = 2673$. The blockmodel induced by $\mathbf{C}_{10}$ is displayed on the left side of Figure 4, and the corresponding adjacency matrix on its right side.

Adjacency matrix presentation can be used for some hundreds of vertices. For larger graphs we can save the idea by introducing a *contextual matrix presentation* of selected block – adjacency matrix for a selected block surrounded by a blockmodel (see Figure 5).

Another, indirect [2], approach to analyze the graph is to introduce a dissimilarity d into set V, or its subset, and apply some of multivariate techniques to it. We applied Ward's clustering method to the dissimilarity d from Section 3. The results are comparable to that obtained by core decomposition and blockmodeling.

5 Conclusion

The cores, because they can be efficiently determined, are one among few concepts that provide us with meaningful decompositions of large graphs. We expect that different approaches to the analysis of large graphs can be built on this basis. (Most) current clustering and blockmodeling methods can be applied only

to (sub)graphs of moderate size – the development of subquadratic algorithms for these problems is a challenge for a near future.

All the computations in this paper were done with programs `Pajek` (Slovene word for Spider) and `Model2` for Windows (32 bit). They are freely available, for noncommercial use, at their homepage [10].

References

1. BATAGELJ, V. (1997): Notes on Blockmodeling. *Social Networks*, **19**, 143-155.
2. BATAGELJ, V., FERLIGOJ, A., and DOREIAN, P. (1992). Direct and Indirect Methods for Structural Equivalence. *Social Networks* **14**, 63-90.
3. BATAGELJ, V., FERLIGOJ, A., and DOREIAN, P. (1998): Fitting Pre-Specified Blockmodels, in *Data Science, Classification, and Related Methods*, Eds., C. Hayashi, N. Ohsumi, K. Yajima, Y. Tanaka, H. H. Bock, and Y. Baba, Springer-Verlag, Tokyo, p.p. 199-206.
4. BATAGELJ, V., FERLIGOJ, A. (1998): Constrained Clustering Problems, in *Advances in Data Science and Classification* Eds., A. Rizzi, M. Vichi, H.-H. Bock, Proceedings of IFCS'98, Rome, 21-24. July 1998. Springer, Berlin, p. 137-144.
5. BATAGELJ, V., MRVAR, A. (1998): Pajek – A Program for Large Network Analysis. *Connections* **21 (2)**, 47-57.
6. FRUCHTERMAN, T. M. J., REINGOLD, E. M. (1991): Graph Drawing by Force-Directed Placement. *Software, Practice and Experience* **21**, 1129-1164.
7. KAMADA, T., KAWAI, S. (1989): An Algorithm for Drawing General Undirected graphs. *Inf. Proc. Letters* **31**, 7-15.
8. SEIDMAN, S. B. (1983): Network structure and minimum degree. *Social Networks* **5**, 269-287.
9. SNYDER, D., KICK, E. (1979): Structural position in the world system and economic growth 1955-70: A multiple network analysis of transnational interactions. *American Journal of Sociology* **84**, 1096-1126.
10. Program Pajek
http://vlado.fmf.uni-lj.si/pub/networks/pajek/

Graph Clustering Using Distance-k Cliques

Software Demonstration

Jubin Edachery[1], Arunabha Sen[1*], and Franz J. Brandenburg[2]

[1] Department of Computer Science
Arizona State University
Tempe 85287, AZ, USA
{jubin,Arunabha.Sen}@asu.edu

[2] Lehrstuhl für Informatik
Universität Passau
94030 Passau, Germany
brandenb@informatik.uni-passau.de

Abstract. Identifying the *natural clusters* of nodes in a graph and treating them as *supernodes* or *metanodes* for a higher level graph (or an abstract graph) is a technique used for the reduction of visual complexity of graphs with a large number of nodes. In this paper we report on the implementation of a clustering algorithm based on the idea of *distance-k cliques*, a generalization of the idea of the *cliques* in graphs. The performance of the clustering algorithm on some large graphs obtained from the archives of Bell Laboratories is presented.

1 Introduction

Visualization tools can be of tremendous help to network planners and administrators for management and control of large networks. For this purpose, a hierarchical view of the network is most appropriate. This provides the network administrators at different levels the ability to view their realm of the network at an appropriate level of detail without being overwhelmed by unnecessary and unimportant minutiae. This hierarchical view of the networks can be provided in graphs by grouping the nodes into some *supernodes* or *metanodes* [3]. As a result, the higher level graph will have much fewer metanodes and thus the visual complexity will be significantly reduced.

Identifying the *natural clusters* of nodes in a graph and treating them as supernodes or metanodes for a higher level graph is a technique that can used for the reduction of visual complexity of graphs with a large number of nodes. An abstract graph is constructed in such a way that a node in the abstract graph represents a set of nodes of the original graph and the edges represent the relationship between these sets of nodes. Thus an abstract graph construction

* Corresponding author, Telephone: 480-965-6153, Fax: 480-965-2751; This research in part was supported by a grant from the NATO Scientific and Environmental Affairs Division and a grant from Tom Sawyer Software under NIST Advanced Technology Program.

J. Kratochvíl (Ed.): GD'99, LNCS 1731, pp. 98–106, 1999.

problem reduces to the problem of partitioning the node set of the original graph $G = (V, E)$, into a subset of nodes $V_1, V_2, ..., V_k$, such that $\cup_{i=1}^{k} V_i = V$ and $V_i \cap V_j = \emptyset$ for $i \neq j$.

The subsets V_is ($1 \leq i \leq k$) are known as the *clusters* of the graph $G = (V, E)$. One problem with this approach is that there is no consensus among the researchers as to what constitutes a *natural cluster*. There is some intuitive understanding of what constitutes a cluster but there is no universally accepted formal definition of a *natural cluster*. In case the nodes and edges of the graph have some semantic information associated with them (in the form of labels), such information can be used for the purpose of clustering (or *grouping*) the nodes. An example of such information could be the IP addresses associated with the nodes in a telecommunication network. In case the graph has no such information, then the structural properties of the graph have to be utilized for the purpose of generating the clusters. Several candidates for the structures to be used as clusters have been proposed in the literature. These include *biconnected components*, *paths and triangles*, *circles of cliques* and others [1, 2, 5, 6].

2 Clustering Using Distance-k Cliques

In spite of the differences of opinion as to what constitutes a cluster, one idea is universally accepted: the nodes belonging to a cluster must have a *strong relationship* between them in comparison with the nodes outside the cluster. Now, we are confronted with another question and that is how to measure the *strength* of a relationship ? In this paper we measure the strength of a relationship between two nodes in a graph in terms of the *distance* between the nodes. The distance between two nodes in a graph is the length of the shortest *path length* between the two nodes. If $dist(u, v)$ gives the distance between the nodes u and v, and if $dist(p, q) > dist(r, s)$, then we say that the strength of the relationship between p and q is *weaker* than the strength of the relationship between r and s. In other words, the strength of a relationship between two nodes is *inversely* proportional to the distance between them.

A subset V' of the node set V of a graph $G = (V, E)$ is defined to be a *Distance-k Clique* if every pair of nodes in V' is connected in G by a path of length at most k. The standard graph theoretic term, *clique*, is a special case of a distance-k clique with $k = 1$. It may be noted that a distance-k clique of a graph $G = (V, E)$ is a subgraph of G with *diameter k*.

In our clustering problem, we would like to partition the node set V of the graph $G = (V, E)$ into fewest number of distance-k cliques. The decision problem version of our clustering problem can be stated as follows:

Partition into Distance-k Cliques Problem
INSTANCE: Given a graph G=(V, E) and positive integers j and k, $1 \leq j, k \leq |V|$.
QUESTION: Can the vertices of G be partitioned into $i \leq j$ disjoint sets $V_1, V_2, \ldots, V_i$ such that, for $1 \leq m \leq i$,the subgraph induced by V_m is a distance-k clique ?

It is not difficult to realize that the *Partition into Distance-k Cliques* problem is NP-Complete because if $k = 1$, it reduces to the *Partition into Cliques* problem, which is known to be NP-Complete [4].

As Partition into Distance-k Clique problem turns out to be an NP-Complete problem, we look for heuristic solutions to the problem that will produce a *good* solution rather than an *optimal* solution. In the following paragraph we describe a class of such algorithms. All these algorithms are variations of one main algorithm which is referred to as the algorithm A1.

2.1 Description of the Algorithms

The clusters are are referred to as $C_1, C_2, \ldots, C_p$ for some integer p. Every node v of the graph $G = (V, E)$ belongs to one cluster $C_i, 1 \le i \le p$. This information is stored in an array called $C[1, .., n]$. If for a node $v \in V$, $C[v] = j$, it implies that the node v is an element of the cluster C_j.

A node v of the graph $G = (V, E)$ is known as an *bridge node* if $v \in C_i, (v, u) \in E$ and $u \notin C_i$, where C_i represent the cluster i.

The *neighbors* of a node $v \in V$ is defined as the nodes that are adjacent to v in the graph $G = (V, E))$, i.e., $N(v) = \{u|(v, u) \in E\}$.

Every bridge node $v \in V$ will have a *cluster-list* associated with it. The cluster-list associated with node v, is the set of all the clusters where the neighbors of v, not in $C[v]$, belong, i.e., $CL(v) = \{C_i|N(v) \cap C_i \neq \emptyset \text{ and } i \neq C[v]\}$.

The *estimated diameters* of the clusters is an upper bound of the diameter of the clusters. They are stored in an array called $EstDia$. $EstDia(C_i)$ gives the estimated diameter of the cluster C_i.

We first describe the Algorithm A1 and then its variations, algorithms A2 through A5. In all the algorithms, the user specifies the value of k for the algorithms to create distance-k cliques.

Algorithm A1

As a first step, A1 constructs an initial clustering of the nodes. The algorithm InitialCluster1 is used for this purpose. InitialCluster1 chooses high degree nodes of the graph and forms clusters around those nodes. The set of initial clusters called $ClusSet$ and it contains the clusters $C_1, C_2, \ldots, C_p$ for some integer p. This procedure also computes the estimated diameter of each cluster in $ClusSet$ and stores in array $EstDia$. The InitialCluster1 procedure also identifies the bridge nodes at this stage of clustering and for each node $v \in V$ determines the cluster in which this node belongs. This information is stored in array C array,

A list of clusters is associated with each bridge node. The bridge node that has the largest number of nodes in its cluster list is used to (tentatively) form a larger cluster ($ComClus$), comprising of the cluster in which the bridge node belongs and all the clusters in its cluster list. If the estimated diameter of the new cluster is less than or equal to k, then the new cluster is made permanent and the cluster list of all the affected bridge nodes are updated. In case the diameter of the new cluster is greater than k, then such a cluster cannot be formed. In this

case, one cluster, ($RemClus$), is removed from the tentative cluster ($ComClus$) so as to reduce the overall diameter and the cluster list of all the affected bridge nodes are updated. The whole process is repeated till the cluster list associated with each bridge node ($CL(i)$) becomes empty.

Algorithm InitialCluster1 (V, E)

begin
  $V' = V; P = 0;$
  while$(V' \neq \emptyset)$
    begin
      $P = P + 1;$
      $m =$ the highest degree node in V';
      $Cluster(P) = \{m\};$
      $V' = V' - \{m\}$
      $\forall (m, n) \in E$ **do**
        begin
          if $n \in V'$ **then**;
            begin
              $Cluster(P) = Cluster(P) \cup \{n\}$
              $V' = V' - \{n\};$
            end
        end
    end
end

Algorithm 1: Generation of Clusters using Distance-k Cliques

begin
  (S1:) Form IntialClusters; (Suppose at this stage there are
    q bridge nodes, $u_1, u_2, \ldots, u_q$.)
  for $i = 1$ to q ;
    begin
      $CL(i) = \{C_j | N(u_i) \in C_j \ and \ j \neq C[u_i]\};$
      $|CL(i)| = \sum_{C_j \in CL(i)} |C_j|;$
      ($|C_j|$ gives the number of nodes in the cluster C_j)
    end
  while(true);
    begin
      If $\forall i, 1 \leq i \leq q$, $CL(i) = \emptyset$ **then EXIT**;
      (S2:) Find u_m such that the $|CL(m)|$ is the *largest* among
      all $CL(i), 1 \leq i \leq q$;
      $Clist = CL(m) \cup \{C(u_m)\};$
      Create a new cluster $ComClus$
      and set $ComClus = \{v | v \in C_i \ and \ C_i \in Clist\};$
      Find the clusters with the largest and second largest estimated
      diameter in the $Clist$ and call them LC_1 and LC_2;

```
            (S3:) EstDia(ComClus) = EstDia(LC1) + EstDia(LC2) + d,
            where d = 1
            if LC1 = C(um) or LC2 = C(um), otherwise d = 2;
            if EstDia(ComClus) ≤ K then
                begin
                ClusSet = ClusSet ∪ {ComClus} − {Ci|Ci ∈ Clist};
                ∀v ∈ ComClus set C[v] = ComClus;
                    for i = 1 to q;
                        begin
                            ∀Cx ∈ Clist if Cx ∈ CL(i)
                            then remove Cx from CL(i);
                            If at least one Cx is removed from CL(i) and
                            if EstDia(ComClus) < K
                            then CL(i) = CL(i) ∪ ComClus;
                        end
                    if EstDia(ComClus) = K then
                        ∀i such that ui ∈ ComClus set CL(i) = ∅ ;
                end
            else
                begin
                    if (LC1 ≠ C(um))then set RemClus = LC1
                        else set RemClus = LC2;
                    if LC1 = C(um) or LC2 = C(um)then
                        begin
                            ∀ui ∈ ComClus
                            set CL(i) = CL(i) − RemClus;
                            ∀ui ∈ RemClus
                            set CL(i) = CL(i) − ComClus;
                        end
                    else
                        CL(m) = CL(m) − RemClus;
                end
        end
    end
```

Algorithm A2

This is essentially same as algorithm A1, except that the step marked S2 in A1, should be replaced by the following step:

(S2:) Find u_m such that the $|CL(m)|$ is the *smallest* among all $CL(i), 1 \leq i \leq q$;

This step allows smaller clusters to grow in parallel, as opposed to the algorithm A1 where one cluster grows to its maximum size before another cluster gets an opportunity for growth in size.

Algorithm A3

This again is essentially same as algorithm A2, except that in step marked S1, the procedure InitialCluster2 should be used for initial clustering instead of the procedure InitialCluster1. In InitialCluster2, each node forms its own cluster and that is taken as the starting point of the algorithm A3. It may be noted that in this case every node $v \in V$ is a bridge node if $degree(v) > 0$.

Algorithm InitialCluster2 $(V, E), (V = \{v_1, \ldots, v_n\})$

begin
  for $i = 1$ to n **do**
    $Cluster(i) = \{v_i\}$;
end

Algorithm A4

This again is essentially same as algorithm A2, except that in step marked S3, instead of estimated diameter of *ComClus*, the *exact* diameter of *ComClus* is computed. This modification gives a higher quality solution at an increased computational cost.

Algorithm A5

This again is essentially same as algorithm A3, except as in algorithm A4, instead of estimated diameter of *ComClus*, the *exact* diameter of *ComClus* is computed. Just like in A4, this modification gives a higher quality solution at an increased computational cost.

2.2 Analysis of Algorithms

We analyze the algorithm A1. The procedure InitialCluster1 can be carried out in $O(n^2)$ time where n is the number of nodes in the graph. The procedure InitialCluster2 can be carried out in $O(n)$ time. The first for-loop can be carried out in $O(n^2)$ time. The while-loop continues till the cluster list associated with each bridge node becomes empty. A bridge node may become a non-bridge node during execution of the algorithm. However, a non-bridge node will never become a bridge node. The number of bridge nodes can be at most n. Inside the while-loop all steps upto and including step marked S3 can be carried out in $O(n)$. The If-statement following S3 is either true or false. If it is true, the statements inside the begin-end block can be carried out in $O(n\Delta)$ time where Δ is the maximum node degree of the graph. If it is false, the statements within the begin-end block can be carried out in $O(n)$. If the begin-end block associated with the true part is executed once, it reduces the number of bridge nodes by at least one. If the begin-end block associated with the false part is executed Δ times, it is guaranteed that the number of bridge nodes will reduce by at least one. Suppose during a particular run, the true part is executed y times and the false part is executed z times. As a result of y time execution of the true part, at least y bridge nodes will cease to be bridge nodes. The remaining $q - y$ bridge nodes must become non-bridge nodes before the while-loop exits. This might require at

Table 1. Performance comparison between five algorithms.

Number of clusters generated by different algorithms								
Graph	No. of Nodes	Diameter	k	A1	A2	A3	A4	A5
Bell Lab 1487	48	9	4	14	14	13	11	5
			5	14	11	7	5	3
			6	13	10	8	4	2
Bell Lab 1572	65	13	5	14	12	15	9	7
			7	12	9	12	7	2
			9	14	7	9	4	2

most $(q-y)\Delta$ executions of the false part of the If-statement. Therefore the total computation in the while-loop will be $O(n) + O(yn\Delta + n(q-y)\Delta) = O(nq\Delta)$. Since the nuber of bridge nodes q and the maximum node degree Δ can be as high as $O(n)$, the worst case complexity of the algorithm will be $O(n^3)$.

3 Experimental Results and Discussions

We tested our implementation of the distance-k clique based clustering algorithm with a wide range of graphs - small, medium and large. We extensively used the Bell Laboratory graph library for testing purpose. This library has a large collection of graphs of wide range of variation in terms of number of nodes, edges and node degrees. The results of the output of our clustering algorithm on Bell Lab graphs 1487 and 1572 with different values of k is presented in table 1. Visualization of the clustering algorithm results is presented in figures 1 through 6. One way to measure the quality of our heuristic algorithm is to find the proximity of the solution produced by it to the optimal solution. We use the following strategy to compute the quality of our solutions: Suppose that the number of clusters generated by heuristic algorithm $A_i, 1 \leq i \leq 5$ for a given graph $G = (V, E))$ and a specified value of k, is $NC_{i,k}$. Suppose for the same graph and same specified k, the optimal number of clusters is OC_k. If we know OC_k, we can easily find its proximity to $NC_{i,k}$. However, OC_k in general is not known. However, we can make the following observation about OC_k: $OC_k \geq diameter(G)/k$. We evaluated the performance of our heuristic algorithms using this criteria. The performance of A_5 is much better than the performance of A_1 at the expense of higher computational time. The performance of the algorithms A1, A2 and A3 is poor in comparison with A4 and A5 because the error in estimated diameter keeps accumulating, resulting in a larger number of clusters. It may be noted that as the objective of the distance-k clustering algorithm is to cluster the nodes that have strong relationship between them, unlike many other clustering algorithms, it does not necessarily minimize the number of intercluster edges.

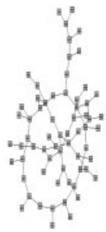


Fig. 1. Bell Lab graph 1572 before clustering

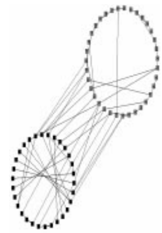


Fig. 2. Bell Lab graph 1572 A5k7

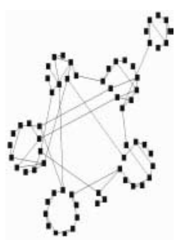


Fig. 3. Bell Lab graph 1572 A4k7

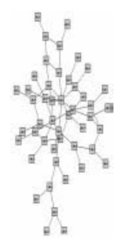


Fig. 4. Bell Lab graph 1487 before clustering

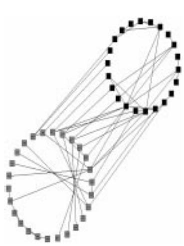


Fig. 5. Bell Lab graph 1487 A5k6

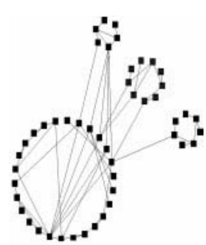


Fig. 6. Bell Lab graph 1487 A4k6

References

[1] F. J. Brandenburg, "Graph Clustering: Circles of Cliques," *Proceedings of Graph Drawing Symposium, GD'97* Rome, September 1997. Lecture Notes in Computer Science, Berlin: Springer-Verlag, 1353, pp. 158-168, 1998.

[2] J.S. Deogun, D. Kratsch and G. Steiner, "An approximation algorithm for clustering graphs with dominating diametral paths," *Information Processing Letters*, 61, pp. 121-127, 1997.

[3] P. Eades, "Multilevel Visualization of Clustered Graphs," *Proceedings of Graph Drawing'96*, Berkeley, California, September,1996.

[4] M.R. Garey and D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman Publishers, San Francisco, 1978.

[5] D.W. Matula and L.L. Beck, "Smallest-last ordering and clustering and graph coloring algorithms," *Journal of the Association of Computing Machinery* 30 pp. 417-427, 1983.

[6] T. Roxborough and A. Sen, "Graph clustering using multiway ratio cut," *Proceedings of Graph Drawing Symposium, GD'97* Rome, September 1997. Lecture Notes in Computer Science, Berlin: Springer-Verlag, 1353, pp. 291-296, 1998.

A Framework for Circular Drawings of Networks*

Janet M. Six and Ioannis G. Tollis

CAD & Visualization Lab
Department of Computer Science
The University of Texas at Dallas
P.O. Box 830688, EC 31
Richardson, TX 75083-0688
{janet,tollis}@utdallas.edu

Abstract. Drawings of graphs which show the inherent strengths and weaknesses of structures with clustered views would be advantageous additions to many network design tools. In this paper we present a framework for producing circular drawings of networks represented by non-biconnected graphs. Furthermore, the drawings produced by these techniques clearly show the biconnectivity structure of the given networks. We also include results of an extensive experimental study which shows our approach to significantly outperform the current state of the art.

1 Introduction

Graphs are used to represent many kinds of information structures: computer, telecommunication, and social networks, entity-relationship diagrams, data flow charts, resource allocation maps, and much more. *Graph Drawing* researchers develop techniques which embed graphs onto a two or three-dimensional surface where nodes are represented by circles or polygons and edges by polygonal chains of line segments. The input to a graph drawing algorithm is a graph, $G = (V, E)$, where V is the set of n nodes and E is the set of m edges, while the output is a set of layout coordinates for each graph element. See [3,4] for a comprehensive annotated bibliography and introduction to the area.

A *circular graph drawing* (see Figure 1 for an example) is an embedding of a graph with the following characteristics: the graph is partitioned into clusters, the nodes of each cluster are placed onto the circumference of an individual embedding circle, and each edge is drawn as a straight line.

Tools for the design and manipulation of telecommunication networks can be strengthened with the addition of a circular drawing component which shows clustered views of those information structures. See [11] for a comprehensive discussion of telecommunication network design algorithms. The partitioning of the graph into clusters can show structural information such as biconnectivity

* Research supported in part by the Texas Advanced Research Program under grant number 009741-040.

J. Kratochvíl (Ed.): GD'99, LNCS 1731, pp. 107–116, 1999.

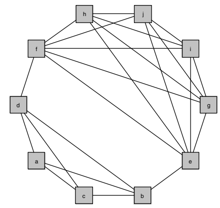


Fig. 1. A circular drawing as produced by our algorithm.

characteristics or can highlight semantic qualities of the network such as subnets. Emphasizing natural group structures within the topology of the network is vital to pin-point strengths and weaknesses within that design. It is essential that the number of crossings within the drawing of each cluster remain low and that nodes have good proximity to their neighbors. Researchers have produced several circular drawing techniques [2,6,10,12,17], some of which have been integrated into commercial tools. The resulting drawings can be compared with each other by counting the number of edge crossings. These drawings are often visually complex with respect to the number of edge crossings. In fact, the problem of producing a circular drawing with a minimum number of crossings was proven to be NP-Complete in [13]. In [15] we present a technique for producing circular drawings of biconnected graphs on a single embedding circle and an experimental study which shows the technique to perform well. A refined version of this approach is discussed in [16]. In this paper, we introduce a framework of efficient techniques to produce circular drawings of nonbiconnected networks. These algorithms require at most $O(m)$ time and have been designed to produce drawings which clearly show biconnectivity.

2 Circular Drawings of Biconnected Graphs

In this section we present a summary of an algorithm for obtaining circular drawings of biconnected networks such that all the nodes are placed onto the circumference of a single embedding circle. In addition, we include the results of an experimental study which shows this technique to be a significant improvement over the current state of the art.

In order to find a circular drawing with a lower number of crossings than previous techniques, we have developed an algorithm which tends to place edges toward the outside of the embedding circle. This characteristic means that there are not many edges in the middle of the drawing to be crossed and also that nodes are placed near their neighbors. In fact, this algorithm tries to maximize the number of edges appearing on the circumference of the embedding circle. This is achieved by selectively removing some edges and then building a DFS-based node ordering of the resulting graph. The number of crossings can then be further reduced with a postprocessing step.

In order to selectively remove some edges, Algorithm *CIRCULAR* visits the nodes in a wave-like fashion. Define a *pair edge* to be incident to two nodes which share at least one neighbor. Each time a node is processed, pair edges induced by the current node are found and placed into a removal list, a "thread" of edges is run through its neighbors to maintain biconnectivity and then the node is deleted. After all of the nodes have been processed, the graph is restored to its original topology and then the removal list edges are removed. It is this selective edge removal that causes the behavior of edge placement towards the perimeter of the embedding circle. Subsequent to the edge removal, *CIRCULAR* conducts a depth-first search (DFS) and places the nodes in a longest path of the DFS tree around the embedding circle. Finally, the remaining nodes are nicely merged into this ordering. The time complexity of *CIRCULAR* is $O(m)$ where m is the number of edges. Please see [15,16] for further details of the algorithm and its analysis. The algorithm has the following interesting property:

Theorem 1. *Given a biconnected graph, G, if G admits a circular layout with zero crossings, then CIRCULAR produces a circular drawing with zero crossings in $O(n)$ time.*

A graph G is *outerplanar* if and only if G can be drawn on the plane such that all nodes lie on the boundary of a single face and no two edges cross [9]. This fact implies that the set of biconnected graphs which admit a plane circular drawing is exactly the set of outerplanar graphs. Given an outerplanar graph, *CIRCULAR* will produce a plane circular drawing since the edge removal portion of the technique is inspired by the algorithm for recognizing outerplanar graphs in [14].

By Theorem 1, if a zero-crossing layout exists for an input biconnected graph, then *CIRCULAR* will find it, however if the input graph does not admit a plane drawing, we offer in [15,16] a heuristic technique to iteratively reduce the number of crossings. We have implemented our technique for building circular drawings of biconnected graphs, *CIRCULAR*, in C++ and run experiments over a set of 10,328 biconnected graphs from [5]. The average number of crossings in the drawings produced by our $O(m)$ time *CIRCULAR* technique is about 15% less than that of the $O(n^2)$ time GLT technique [6,10]. Performing the postprocessing step improves upon the average number of crossings of GLT drawings by 30%.

3 Circular Drawings of Nonbiconnected Graphs

Graphs are not always biconnected, therefore it is important for a graph drawing tool to provide a component which visualizes nonbiconnected graphs. In this section we present techniques for drawing nonbiconnected graphs. Furthermore, these drawings will clearly show the biconnected components.

3.1 Drawings of Nonbiconnected Graphs

Given a nonbiconnected graph, G, we can decompose the structure into biconnected components in $O(m)$ time. By definition, the biconnected components will

be connected via bridges or articulation points and this superstructure will form a *block-cut point tree.* Taking advantage of this inherent structure, we first layout the biconnected components of the block-cut point tree with a radial layout technique similar to [1,7,8], then layout each biconnected component of the network with a variant of the *CIRCULAR* algorithm. See Figure 2. This technique is called *CIRCULAR - with Radial.*

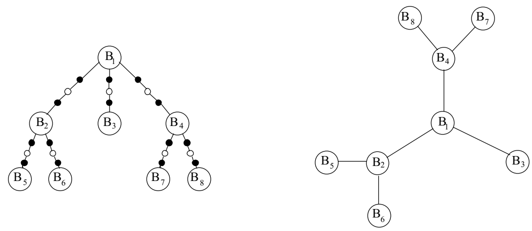


Fig. 2. The illustration on the left shows the block-cut point tree of a non-biconnected graph. The small black tree nodes represent articulation points and the small white tree nodes represent bridges. The right illustration is a drawing of the same graph where the block-cut point tree is laid out with a radial tree layout technique.

CIRCULAR - with Radial must address several issues in order to produce good quality circular drawings: articulation points can appear in multiple biconnected components of the block-cut point tree, the nodes of the block-cut point tree can represent biconnected components of differing size, and the nodes of each biconnected component should be visualized such that the articulation points appear in good position and also there is a low number of edge crossings. We will address each of these issues in turn.

Assignment of Articulation Points: Define a **strict articulation point** as an articulation point which is not adjacent to a bridge. Strict articulation points are duplicated in more than one biconnected component of the block-cut point tree, but of course each node should appear only once in a drawing of that graph. Therefore, we offer three approaches in which the articulation point will appear only once in the drawing. The first approach assigns each strict articulation point to the biconnected component which is closest to the root in the block-cut point tree. This biconnected component will be the parent of the other biconnected components, see Figure 3(a). The second approach assigns the articulation point to the biconnected component which contains the most neighbors of that articulation point, see Figure 3(b). The third approach assigns the articulation point to a position between its biconnected components, see Figure 3(c). Placing a node in this manner will highlight the fact that this node is an important articulation point. Following the assignment step, the duplicates of a strict articulation point are removed from the blocks in the block-cut point tree. We refer to the nodes adjacent to a removed strict articulation point in a

biconnected component as *inter-block nodes*. In order to maintain biconnectivity for the method which will layout this biconnected component, a thread is run through the inter-block nodes. Due to space constraints, further discussion of articulation point assignment is not given here.

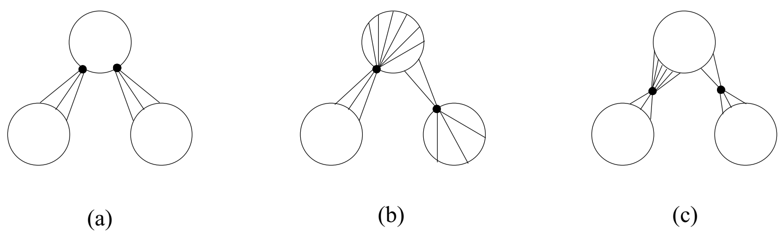


Fig. 3. This figure shows examples of three approaches for the assignment of strict articulation points to biconnected components. The black nodes are strict articulation points.

Radial Layout of Trees With Differing Node Sizes: While performing the layout of the block-cut point tree, we must consider that the biconnected components may be of differing sizes. The radial layout algorithms presented in [1,7,8] place the root at $(0,0)$ and the subtrees on concentric circles around the origin. Also see [4] for a summary. These algorithms require linear time and produce plane drawings. However, unlike our block-cut point trees, the nodes of the trees laid out with [1,7,8] are all the same size. In order to produce radial drawings of trees with differing node sizes, we introduce *RADIAL - with Different Node Sizes*.

For each node, *RADIAL - with Different Node Sizes* must assign a ρ coordinate, which is the distance from point $(0,0)$ to the placement of that node and a θ coordinate which is the angle between the line from $(0,0)$ to $(\infty,0)$ and the line from $(0,0)$ to the placement of that node. The ρ coordinate of node v, $\rho(v)$, is defined to be $\rho(u)+\delta+\frac{d_u}{2}+\frac{max(d_1,d_2,...,d_k)}{2}$, where $\rho(u)$ is the ρ coordinate of the parent u of v, δ is the minimum distance allowed between two nodes, d_u is the diameter of u, and $max(d_1,d_2,...,d_k)$ is the maximum of the diameters of all the children of u. In order to prevent edge crossings, each subtree must be placed inside a wedge, and the width of each wedge must be restricted such that it does not overlap a wedge of any other subtree. The θ coordinate of node v depends on the widths of the descendants of v, not just the number of leaves as in [1,7,8]. This assignment of coordinates leads to a layout of the form shown in Figure 4.

Circular Layout of a Biconnected Component: After performing *RADIAL - with Different Node Sizes* we have a layout of the block-cut point tree, and need to visualize the nodes and edges of each biconnected component. The radial layout of the block-cut point tree should be considered while drawing each

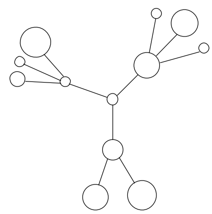


Fig. 4. This illustration demonstrates a radial layout of a tree with differing size nodes.

biconnected component, see Figure 5. In order to reduce the number of crossings caused by inter-biconnected component edges, *CIRCULAR - with Radial* tries to place nodes which are adjacent to the parent biconnected component (in the block-cut point tree) (these will be referred to as *ancestor nodes*) in the arc between α and β and the nodes which are adjacent to child biconnected components (*descendant nodes*) at points γ, δ, ϵ, etc. The size of the arc from α to β is dependent on the distance of the biconnected component to the parent in the radial layout of the block-cut point tree. The points γ, δ, and ϵ are placed in the bottom half of the biconnected component layout. These special positions for the ancestor and descendant nodes are called *ideal positions.* Placing the articulation points and inter-block nodes in this manner reduce the number of crossings caused by inter-biconnected component edges going through a biconnected component. In fact, the only times that these edges do cause crossings with this approach are when the number of ancestor or descendant nodes in the biconnected component, B_i is more than about $\frac{n_i}{2}$, where n_i is the number of nodes in B_i. In those cases, the set of ideal positions includes all the positions in the upper (respectively lower) half of the embedding circle and also positions in the lower(upper) half which are as close as possible to the upper(lower) half.

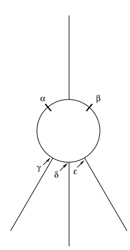


Fig. 5. This figure shows the relation between the layout of the block-cut point tree and the layout of an individual biconnected component.

We present two techniques for the layout of each biconnected component such that ideal positions of the ancestor and descendant nodes are considered. The first step of each technique is to perform *CIRCULAR* on the current biconnected component, B_i. This requires $O(m_i)$ time, where m_i is the number of edges in biconnected component B_i. Then we update this drawing such that the node placement of the articulation points is considered.

The first technique, *CLUSTER1*, rotates the layout of the biconnected component as found by *CIRCULAR* such that a maximum number of ancestor and descendant nodes are placed close to their ideal positions. Then, the remaining ancestor and descendant nodes are moved to the closest ideal position. Algorithm *Cluster1* requires $O(m_i)$ time. See Figure 6(b) for an example.

The second technique, *Cluster2* is an alternative technique which may lead to layouts with fewer edge crossings. The first steps of Algorithm *Cluster2* are the same as that of *Cluster1*. During the placement of ancestor and descendant nodes which are not in ideal positions, each such node, v, is placed in an ideal position and if the number of edge crossings added exceeds a threshold T_1 or the movement of v exceeds a threshold T_2, then the size of the embedding circle is increased such that node v can be placed in an ideal position without changing the relative order between v and its neighbors on the embedding circle. See Figure 6(c) for an example. The time required for Algorithm *Cluster2* is $O(m_i)$ if the threshold T_2 (based on node movement) is used or $O(m_i * k)$, where k is the number of ancestor and descendant nodes in the cluster, if the threshold T_1 (based on the number of crossings) is used.

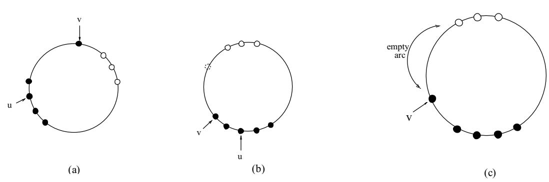


Fig. 6. This figure demonstrates Algorithms *Cluster1* and *Cluster2*. The black nodes are descendant nodes and the white nodes are ancestor nodes. (a) drawing produced by *CIRCULAR*; (b) the rotated drawing of part (a) produced by *Cluster1*. (c) the resulting drawing of part (a) produced by *Cluster2*.

3.2 Algorithm CIRCULAR - with Radial

Now we present a comprehensive technique for obtaining circular layouts of nonbiconnected graphs. First the input graph, G, is decomposed into a block-cut point tree, T. If G is not biconnected, then we layout the root cluster with *CIRCULAR* in order to produce a drawing of the root cluster with a low number of edge crossings. Next a radial layout of each subtree of the root cluster is

placed in previously computed wedges around that cluster. See [16] for more details. Then drawings of each biconnected component are produced. Finally, the biconnected component drawings are translated to their final coordinates according to the radial layout of their parent subtrees.

Algorithm 31 *Algorithm* **CIRCULAR - with Radial**

Input: Any graph, G.
Output: A circular drawing of G.

1. Decompose G into a block-cut point tree, T.
2. If G has only one biconnected component
3. Perform *CIRCULAR* on G.
4. Else
5. Assign the strict articulation points to a biconnected component.
6. Layout the root cluster of T with *CIRCULAR*.
7. For each subtree, S, of the root cluster
8. Perform the ρ coordinate assignment phase of *RADIAL - with Different Node Sizes* on S.
9. For each biconnected component, B_i, of S
10. Layout B_i with *Cluster1* or *Cluster2* taking into account the radii defined for the superstructure tree in Step 8.
11. Considering the order of the subtrees defined during the layout of biconnected components in Step 10, perform the θ coordinate assignment phase of *RADIAL - with Different Node Sizes* on S.
12. Translate and rotate the clusters of S according to the radial layout of S.

The time complexity of *CIRCULAR - with Radial* is $O(m)$ if the biconnected components are laid out with *Cluster1* or $O(m * k)$, where k is the total number of ancestor and descendant nodes in the graph, if *Cluster2* is the algorithm chosen. Regardless of the choice of these two algorithms for the layout of each biconnected component, the biconnectivity structure is clearly shown in the drawings produced by *CIRCULAR - with Radial* since the block-cut point tree superstructure is laid out with a tree layout method that prevents the drawings of any two biconnected components from overlapping. Furthermore, the layout of each biconnected component is based on a technique which has been shown to perform very well with respect to producing drawings with a low number of crossings.

3.3 Implementation and Experiments

We have implemented *CIRCULAR - with RADIAL* with postprocessing in C++ and run preliminary experiments with 11,399 graphs from [5]. The plot in Figure 7 shows the average number of edge crossings produced by the circular

layout component of Tom Sawyer Software's **Graph Layout Toolkit** (GLT), and *CIRCULAR - with Radial*. As is shown by these results, the average number of crossings in the drawings produced by our technique is about 35% less than that of the GLT technique [6,10]. Sample drawings from both the GLT and *CIRCULAR - with Radial* are shown in Figure 8.

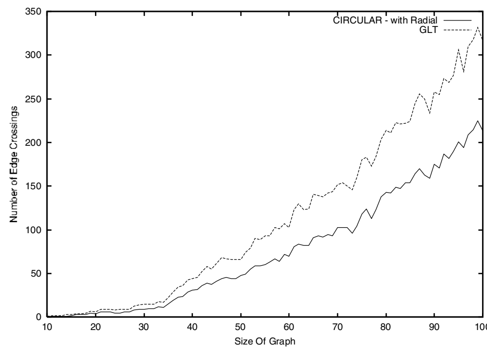


Fig. 7. This plot shows the average number of edge crossings produced by *CIRCULAR* and the Graph Layout Toolkit over 11,399 graphs from [5].

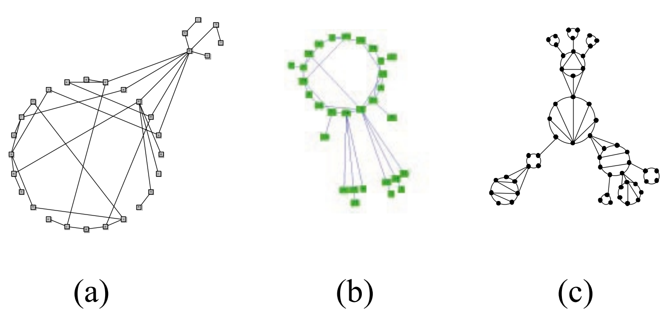


Fig. 8. Example output from the GLT and *CIRCULAR - with RADIAL*. (a) one of the graphs from [5] as drawn with the GLT; (b) drawing of the graph in (a) as produced by *CIRCULAR - with RADIAL*; (c) a sample drawing from *CIRCULAR - with RADIAL*.

Acknowledgements

We would like to thank Tom Kurien and Yong "Bruce" Zhao for helpful discussions and their skilled implementation of software components for this project.

References

1. M. A. Bernard, On the Automated Drawing of Graphs, *Proc. 3rd Caribbean Conf. on Combinatorics and Computing*, pp. 43-55, 1994.
2. F. Brandenburg, Graph Clustering 1: Cycles of Cliques, *Proc. GD '97, Rome, Italy, Lecture Notes in Computer Science 1353, Springer-Verlag*, pp. 158-168, 1998.
3. G. Di Battista, P. Eades, R. Tamassia and I. Tollis, Algorithms for Drawing Graphs: An Annotated Bibliography, *Computational Geometry: Theory and Applications*, 4(5), pp. 235-282, 1994. Also available at `http://www.utdallas.edu/~tollis`.
4. G. Di Battista, P. Eades, R. Tamassia and I. Tollis, *Graph Drawing: Algorithms for the Visualization of Graphs*, Prentice-Hall, Englewood Cliffs, NJ, 1999.
5. G. Di Battista, A. Garg, G. Liotta, R. Tamassia, E. Tassinari, F. Vargiu and L. Vismara, An Experimental Comparison of Four Graph Drawing Algorithms, *Computational Geometry: Theory and Applications*, 7(5-6), pp.303-26, 1997. Also available at `http://www.cs.brown.edu/people/rt`.
6. U. Doğrusöz, B. Madden and P. Madden, Circular Layout in the Graph Layout Toolkit, *Proc. GD '96, Berkeley, California, Lecture Notes in Computer Science 1190, Springer-Verlag*, pp. 92-100, 1997.
7. P. D. Eades, Drawing Free Trees, *Bulletin of the Institute for Combinatorics and its Applications*, 5, pp. 10-36, 1992.
8. C. Esposito, Graph Graphics: Theory and Practice, *Comput. Math. Appl.*, 15(4), pp.247-253, 1988.
9. F. Harary, *Graph Theory*, Addison-Wesley, Reading, MA, 1969.
10. G. Kar, B. Madden and R. Gilbert, Heuristic Layout Algorithms for Network Presentation Services, *IEEE Network*, 11, pp. 29-36, 1988.
11. A. Kershenbaum, *Telecommunications Network Design Algorithms*, McGraw-Hill, 1993.
12. V. Krebs, Visualizing Human Networks, *Release 1.0: Esther Dyson's Monthly Report*, pp. 1-25, February 12, 1996.
13. S. Masuda, T. Kashiwabara, K. Nakajima and T. Fujisawa, On the NP-Completeness of a Computer Network Layout Problem, *Proc. IEEE 1987 International Symposium on Circuits and Systems, Philadelphia, PA*, pp.292-295, 1987.
14. S. Mitchell, Linear Algorithms to Recognize Outerplanar and Maximal Outerplanar Graphs, *Information Processing Letters*, 9(5), pp. 229-232, 1979.
15. J. M. Six and I. G. Tollis, Circular Drawings of Biconnected Graphs, *Proc. of ALENEX '99, Baltimore, MD*, To appear, 1999.
16. J. M. Six and I. G. Tollis, Algorithms for Drawing Circular Visualizations of Networks, Manuscript, 1999.
17. I. G. Tollis and C. Xia, Drawing Telecommunication Networks, *Proc. GD '94, Princeton. New Jersey, Lecture Notes in Computer Science 894, Springer-Verlag*, pp. 206-217, 1994.

Drawing Planar Graphs with Circular Arcs

C. C. Cheng*, C. A. Duncan** ***, M. T. Goodrich***, and S. G. Kobourov***

The Johns Hopkins University
Baltimore, MD 21218

Abstract. In this paper we address the problem of drawing planar graphs with circular arcs while maintaining good angular resolution and small drawing area. We present a lower bound on the area of drawings in which edges are drawn using exactly one circular arc. We also give an algorithm for drawing n-vertex planar graphs such that the edges are sequences of two continuous circular arcs. The algorithm runs in $O(n)$ time and embeds the graph on the $O(n) \times O(n)$ grid, while maintaining $\Theta(1/d(v))$ angular resolution, where $d(v)$ is the degree of vertex v. Since in this case we use circular arcs of infinite radius, this is also the first algorithm to simultaneously achieve good angular resolution, small area and at most one bend per edge using straight-line segments. Finally, we show how to create drawings in which edges are smooth C^1-continuous curves, represented by a sequence of at most three circular arcs.

1 Introduction

The study of methods for rendering planar graphs is central in the graph drawing literature. In planar graph drawings, vertices are represented by distinct points in the plane and edges are drawn as continuous curves that do not cross one another [1]. An important characteristic of a graph drawing is its readability, and some of the essential qualities that determine readability include the following:

1. *edge smoothness:* edges should be drawn with "smooth" curves. Ideally, we prefer straight line segments. If some other considerations prevent the use of straight lines, then edges should be drawn as simple smooth low-degree curves or polylines with few bends.
2. *area:* vertices and bend points should be placed at integer grid points in as small a box as possible. Ideally, vertices and bend points should be placed on an $O(n) \times O(n)$ grid, where n is the number of vertices in the graph.
3. *angular resolution*: for each pair s and t of curves representing two consecutive edges incident on a vertex v, the angle between the tangent of s at v and the tangent of t at v should be large. Ideally, we would like the measure of each such angle to be $\Theta(1/d(v))$, where $d(v)$ denotes the degree of v.

* Partially supported by ONR grant N00014-96-1-0829.
** Also at Max-Planck-Institut für Informatik.
*** Partially supported by NSF grant CCR-9732300 and ARO grant DAAH04-96-1-0013.

J. Kratochvíl (Ed.): GD'99, LNCS 1731, pp. 117–126, 1999.

Thus, we are interested in a study of methods for drawing planar graphs with smooth edges, small area, and ideal angular resolution. The particular emphasis in this paper is to consider methods for drawing edges with polylines such that each piece of the polyline is drawn with a circular arc. This is a strict generalization of the usual piecewise-linear polylines [8,10], since a straight line segment can be viewed as an arc of a circle of infinite radius. In this paper we address the following questions: What area is achievable for drawings with good angular resolution that use single circle arcs for edges? What area is achievable for drawings that use at most two circular arcs per edge and have good angular resolution? What is the fewest number of circular arcs needed to achieve $O(n) \times O(n)$ area, good angular resolution, and C^1-continuity for edges?

1.1 Prior Related Work

There is a rich body of knowledge that has been developed for drawing planar graphs. Early work by Wagner [14], Fary [4], and Tutte [13] focused on drawings of planar graphs using straight line edges, without much attention paid to other aesthetic or complexity issues. Indeed, the drawings produced using these early techniques can in many cases require exponential area. Later de Fraysseix *et al.* [3] and then Schnyder [12] showed that one can draw a planar graph with straight line edges and vertices placed at grid points in an $O(n) \times O(n)$ integer grid. Still, the drawings produced from these algorithms have a weakness, which is not as prevalent in the algorithms based on Tutte's approach: namely, the area-efficient straight-line drawings can produce very small angles between consecutive edges incident upon the same vertex (poor angular resolution). In fact, it has been proven [11] that there exist graphs that always require exponential area for straight-line embeddings maintaining good angular resolution.

The problem of drawing planar graphs with good angular resolution was addressed by Formann *et al.* [5], Garg and Tamassia [6], and Kant [9,10], who showed that one could in fact simultaneously achieve $O(n) \times O(n)$ area and an angular resolution of $\Theta(1/d(v))$ for each vertex v, by drawing a planar graph using piecewise linear polylines with at most three bends each. Gutwenger and Mutzel [8] improved the constant factors for such drawings, establishing that one could draw an n-vertex planar graph in a $(2n-5) \times (3n/2-7/2)$ grid with at least $2/d_{\mathtt{max}}$ angular resolution using piecewise linear polylines with at most three bends each, where $d_{\mathtt{max}}$ is the maximum degree of the graph. Goodrich and Wagner [7] showed that one could in fact achieve $O(n) \times O(n)$ area with an angular resolution of $\Theta(1/d(v))$ for each vertex v, using piecewise linear polylines with only two bends each. They also showed that one could achieve the same area and angular resolution bounds using smooth degree-3 (Bézier) curves.

1.2 Our Results

In this paper we provide answers to the questions posed above. Specifically, we show the following:

- There exists an n-vertex planar graph G that requires area exponential in n for any drawing of G that uses single circle arcs for edges and has good angular resolution.
- We can draw an n-vertex planar graph G in an $O(n)\times O(n)$ grid with angular resolution $\Theta(1/d(v))$ for each vertex v in G using at most two circular arcs per edge. In fact, in this case we use circular arcs of infinite radius so that the polylines are piecewise linear with at most one bend each, while still maintaining good angular resolution and $O(n) \times O(n)$ area.
- We can draw an n-vertex planar graph G in an $O(n)\times O(n)$ grid with angular resolution $\Theta(1/d(v))$ for each vertex v in G using C^1-continuous curves that consist of at most three circular arcs.

Our lower-bound proof is based on a non-trivial analysis of a circular-arc drawing of the well-known nested-triangles graph. Our algorithm is based on a careful modification of the incremental approach to planar graph drawing utilized by de Fraysseix *et al.* [3] similar to the approach used by Goodrich and Wagner [7]. We describe the main ideas behind these results in the sections that follow, beginning our discussion with the algorithm.

2 Algorithm

We now describe an efficient algorithm, `OneBend`, to embed any planar graph on an $O(n) \times O(n)$ grid while maintaining good angular resolution, $\Theta(1/d(v))$, for each vertex v, and using at most one bend per edge. Following the methods of de Fraysseix *et al.* [3] and Kant [10], we insert vertices sequentially by their canonical ordering, generating subgraphs $G_1, G_2, \ldots, G_n$ in the process. Recall that in the canonical order, vertices are labeled $v_1, v_2, \ldots, v_n$ and graph G_i is defined to be the subgraph induced on the vertices $v_1, v_2, \ldots, v_i$. Graph G_i is 2-connected and its external face is a cycle C_i. Furthermore, in graph G_{i+1}, the new vertex, v_{i+1} has all of its neighbors on the external face of C_i.

In the manner of Goodrich and Wagner [7], we use a box around each vertex of size proportional to its degree but guarantee that each edge drawn contains at most one bend rather than the previous best known method using two. To generate a subgraph G_{k+1} from G_k by inserting a vertex v_{k+1} and its associated box, we will need to perform a few operations and maintain a few sets. Let $w_1 = v_1, w_2, \ldots, w_m = v_2$ be the vertices of the exterior face C_k of G_k in order. For a particular subgraph G_k and vertex v_{k+1}, we refer to w_l and w_r as the leftmost and rightmost neighbors of v_{k+1} on C_k; see Figure 1(a).

2.1 Vertex Joint Box

We associate with every vertex $v \in V$ a *joint box* centered around v, rotated 45°, and having width and height $4d(v) + 4$ units, see Figure 1(b). For notational convenience, if v is clear from the context, then we will use d to denote the degree, $d(v)$, of v. Thus, if v is located at position (i, j), the four corners of the

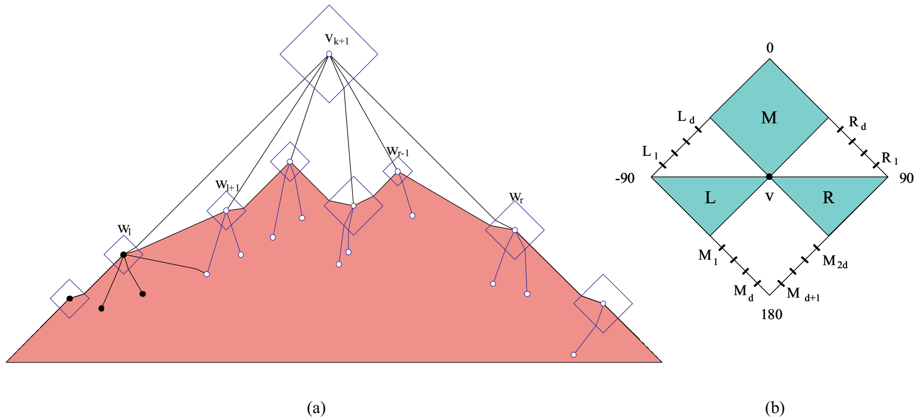


Fig. 1. (a) Graph G_{k+1} after inserting v_{k+1}. The shaded part is G_k and all unfilled vertices are part of the shifting set $M_{k+1}(v_{k+1})$. (b) The **joint** box for a vertex, v.

box are $(i \pm 2d + 2, i)$ and $(i, j \pm 2d + 2)$. We break the box into two types of alternating regions, *free regions* and *port regions*. For each free region there is at most one edge passing through it to v. Each port region consists of a collection of d ports and every edge inside the port region passes through a unique port. Define the free regions using angular coordinates clockwise around v as follows:

- Free region M lies between -45° and 45°.
- Free region R lies between 90° and 135°
- Free region L lies between -135° and -90°.

In between each of these regions are the port regions. For reference, we label the ports between L and M upward as $L_1, \ldots L_d$ and similarly between R and M. The ports between L and R are labelled M_1, M_{2d} in counterclockwise order.

The algorithm draws each edge in E by "routing" it through a port in the joint box of one of the two vertices. Each edge consists of two connected edge segments. The first edge segment, the *port edge segment*, connects a vertex with one of its ports while the second segment, the *free edge segment*, connects the vertex to one of its neighbor's ports.

2.2 The Invariants

In order to incrementally construct our embedding, we maintain invariants similar to those of de Fraysseix *et al.* [3] and Goodrich and Wagner [7] with two important differences (a slight change in invariant three and a new invariant four):

1. The vertices and the ports of the joint boxes have integer coordinates.
2. Let $w_1 = v_1, w_2, \ldots, w_m = v_2$ be the vertices on the exterior face C_k of G_k in order. Then $x(w_1) < x(w_2) < \cdots < x(w_m)$, where $x(w_i)$ is w_i's x-coordinate.

3. Edge (w_i, w_{i+1}), for $0 < i < m$ has the free edge segment with slope ± 1.
4. For every vertex v there is at most one edge segment crossing each of its free regions. All other edge segments are port edge segments.

Notice that if invariant four holds for the embedding G_k, by the definition of the joint box and location of the port regions, G_k has angular resolution no worse than $\Theta(1/d(v))$, for each vertex v.

2.3 The Shifting Set

During each insertion, we must create space for the vertex joint box to "see" its leftmost and rightmost neighbors without the box touching any of the neighbors along the face in between. To do this, we need to shift the vertices along the external face by a certain amount. However, in order for the invariants and planarity to be guaranteed other vertices must also be shifted at the same time. As in de Fraysseix *et al.* [3] and Goodrich and Wagner [7], we define the shifting set for a vertex w_i on the external face of G_k as $M_k(w_i)$. For any graph G_k, we define $M_k(w_i) \subseteq V$ so that the following conditions hold:

1. $w_j \in M_k(w_i)$ if and only if $j \geq i$.
2. $M_k(w_1) \supset M_k(w_2) \supset \cdots \supset M_k(w_m)$.
3. For any nonnegative numbers $\delta_1, \delta_2, \ldots, \delta_m$, if we sequentially translate all vertices in $M_k(w_i)$ with distance δ_i to the right ($i = 1, 2, \ldots, m$), then the embedding of G_k remains planar.[1]

Recall that for a vertex $v = v_{k+1}$, w_l and w_r are the leftmost and rightmost neighbors of v on C_k. Starting with the initial shifting set at $k = 3$, we construct $M_{k+1}(w_i)$ recursively as follows: $M_{k+1}(w_i) = M_k(w_i) \cup v_{k+1}, M_{k+1}(v_{k+1}) = M_k(w_{l+1}) \cup v_{k+1}, M_{k+1}(w_j) = M_k(w_j)$, for $i \leq l$ and $j \geq r$.

This construction allows us to guarantee that the above three conditions of the shifting sets are maintained. Intuitively, after a vertex w_i is removed from the external face by another vertex v_{k+1}, it always shifts exactly with v_{k+1}. During any shift, vertices can only get farther apart in the x-direction. Note that in our algorithm, when a vertex is shifted, its joint box is also shifted, i.e., the ports move as well.

2.4 The Construction

We now show how algorithm `OneBend` iteratively constructs graphs $G_1, G_2, \ldots, G_n$. It is trivial to construct the initial cases of G_1, G_2, and G_3, i.e. inserting the first three vertices. Suppose we have embedded G_k with exterior face C_k. Let $C_k = (v_1 = w_1, w_2, \ldots, w_m = v_2)$ be the exterior face of G_k. To construct G_{k+1}, let $v = v_{k+1}$ be the next vertex in the canonical ordering and recall that w_l and w_r are, respectively, the leftmost and rightmost neighbors of v on the face C_k.

[1] Note that many vertices will move several times; e.g. all points in $M_k(w_i) \backslash M_k(w_{i+1})$ will be translated by $\delta_1 + \delta_2 + \cdots + \delta_i$.

Let d, d_l, d_r be the respective degrees of v, w_l, and w_r. Let p_l be the first unused R_i port in w_l's joint box. Similarly, let p_r be the first unused L_i port in w_r's joint box. Recall since each port region has at least d ports available there is always an unused port.

We insert v by shifting all vertices in the shifting set $M_k(w_{l+1})$ by $2d+2$ positions to the right. Additionally we shift all vertices in $M_k(w_r)$ by an additional $2d+2$ positions to the right. This implies all vertices in $M_k(w_r)$ actually move $4d+4$ positions. Finally, we place v at the intersection of lines l and r where l (respectively r) is the line through p_l (respectively p_r) with slope $+1$ (respectively -1). We route the edges between v and w_l through p_l and do the same for w_r. To maintain invariants one and three, notice that if the intersection point has integer coordinates these two invariants hold. Otherwise, by shifting $M_k(w_r)$ one additional unit, the intersection point has integer coordinates.

To complete the insertion and the algorithm, we need to draw the edges between v and w_i, where $l < i < r$. Let w_j be the rightmost vertex with an x-coordinate less than v. We route the edges from v to vertices w_i, where $l < i \leq j$ through consecutive increasing ports from M_1 in v's joint box. Similarly, we route the edges from v to vertices w_i, where $r > i > j$ through consecutive decreasing ports from M_{2d} in v's joint box.

Lemma 1. *After shifting, free edge segments remain in their free regions.*

Proof Sketch: Consider the free edge segments in the M regions. Notice that these segments are created by a vertex v dominating another vertex w. In this case, w joins v's shifting set and is only shifted when v is shifted. Therefore, the slope remains constant and the free edge segment remains within M.

Consider the case when the free edge segment lies in the L region. This implies that the slope of the line is between 0 and $+1$. Since shifting only moves vertices farther apart in the x-direction, the slope can only get closer to 0 while still remaining in L. The argument is similar for R. $\square$

Lemma 2. *After insertion, every free edge segment passes through a free region which contains no other segment.*

Proof Sketch: After inserting a vertex v_{k+1}, there are two types of edges added: edges between v_{k+1} and the outside neighbors, w_l and w_r, and edges between v_{k+1} and w_i where $l < i < r$. In both cases the edge is routed through a port creating one free edge segment and one port edge segment. By construction, a free edge segment of the first type has slope either $+1$ or -1 and so it lies inside v_{k+1}'s joint box free region L or R, respectively. Since v_{k+1} is a new vertex, there are no other segments inside these two free regions.

A free edge segment of the second type intersects the M region of w_i's joint box. Since this can happen at most once, as the vertex is now no longer on an external face, there can be no other free edge segment inside this free region. $\square$

Lemma 3. *If invariants 1-4 hold for G_k, then they also hold for G_{k+1}.*

Proof Sketch: By the nature of the shifting set, invariants one and two hold (see [7]). Since shifting a vertex involves shifting the entire joint box simultaneously, after every shift operation all port edge segments have unchanged slope. Also, after the two shifting operations, all free edge segments on the exterior face have unchanged, albeit ± 1 slope, except possibly the free edge segments (w_l, w_{l+1}) and (w_{r-1}, w_r). However, after insertion, these free edge segments are no longer on the exterior face and are instead replaced by two free edge segments between (w_l, v) and (v, w_r) with slope ± 1. Therefore, invariant three holds.

By lemmas 1 and 2 and the fact that port edge segments never change slope, we see that invariant four also holds since all edges routed in algorithm `OneBend` created a port segment and a free edge segment. □

Theorem 1. *Given a planar graph G, algorithm* `OneBend` *produces in $O(n)$ time a planar embedding on the $30n \times 15n$ grid with angular resolution $\Theta(1/d(v))$ and using any of the following types of edges: polylines with one bend, or two circular arcs with C^0-continuity and one knot, or three circular arcs with C^1-continuity.*

Proof Sketch: The original algorithm as stated produces polylines with one bend per edge. This by definition can also be represented by two circular arcs, straight lines, which have a discontinuity at the bend, or knot. Since the points are embedded on the grid, the bends may also be replaced by circular arcs of a relatively small size to ensure C^1-continuity as well.

It has been shown by Chrobak and Payne [2] how to implement the algorithm of De Fraysseix, *et al.* [3] in linear time. Their approach can be easily extended to our algorithm. It remains to show that the drawings produced by algorithm `OneBend` fit on the $30n \times 15n$ grid. Every time we insert a vertex v_k, we increase the grid size by $4d(v_k) + 5$ units. Summing over all the degrees of the vertices we get $\sum_{v \in V} 4d(v) + 5 = 4(6n - 12) + 5n < 30n$. The final drawing fits inside an isosceles triangle with sides of slope $0, +1, -1$. The width of the base is $30n$ and so the height is less than $15n$. □

3 Drawing with Circular Arcs

Malitz and Papakostas [11] showed that some planar graphs, drawn with straight lines in the $O(n) \times O(n)$ grid must have small angles. More specifically, they found a class of planar graphs, $\mathcal{H}$, whose straight-line planar drawings require exponential area if the angular resolution is good. Suppose we relax the condition that each edge in a graph be drawn with a straight line segment so that each edge is drawn with a circular arc (where a straight line segment is considered an arc from a circle of radius infinity). Can we draw the graphs in $\mathcal{H}$ with angular resolution $\alpha > 0$ in an $O(n) \times O(n)$ grid? Surprisingly, as long as α is a constant, the answer is no.

Let $\mathcal{H} = \{H_n, n \geq 1\}$ and H_1 be a cycle on 3 vertices P_1, Q_1 and R_1. For $n \geq 2$, the graph H_n is constructed from H_{n-1} by adding a cycle on three new vertices P_n, Q_n, R_n, and edges $(P_n, P_{n-1}), (Q_n, Q_{n-1}), (R_n, R_{n-1})$ and

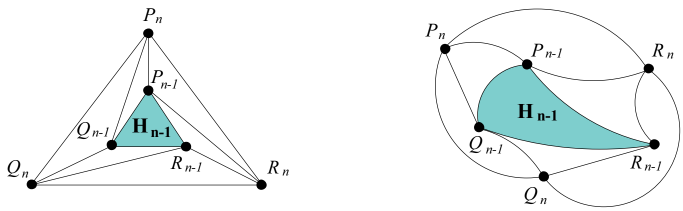


Fig. 2. Graph H_n is constructed from H_{n-1} by adding vertices P_n, Q_n, R_n and the edges shown above. The figure on the right shows H_n drawn with circular arcs.

$(P_n, Q_{n-1}), (Q_n, R_{n-1}), (R_n, P_{n-1})$, as shown in Figure 2. It is easy to check that the graph is planar, triconnected and thus, has a unique embedding. We show that for any planar, circular-arc drawing of H_n with angular resolution $\alpha > 0$, there exists a constant $c_\alpha > 1$ such that the area of the drawing is $\Omega(c_\alpha^n)$.

Let Γ_n be a planar circular-arc drawing of H_n with angular resolution $0 < \alpha \leq \pi/3$. If (u, v) is an edge in H_n then we shall refer to the arc that represents (u, v) in Γ_n as $\widehat{uv}$, and the line segment that connects u and v as $\overline{uv}$. (Sometimes u or v may not be a vertex of H_n but a point on some arc of Γ_n. In this case, $\widehat{uv}$ refers to the portion of the arc that starts at u and ends at v.) If $\mathcal{S}$ is a set of arcs in Γ_n that bounds a region, then we let $Area(\mathcal{S})$ be its area.

Define regions $\mathcal{S}_1$, $\mathcal{S}_2$ and $\mathcal{S}_3$ as follows: $\mathcal{S}_1 = \{\widehat{P_{n-1}Q_{n-1}}, \widehat{Q_{n-1}P_n}, \widehat{P_nP_{n-1}}\}$, $\mathcal{S}_2 = \{\widehat{Q_{n-1}R_{n-1}}, \widehat{R_{n-1}Q_n}, \widehat{Q_nQ_{n-1}}\}$ and $\mathcal{S}_3 = \{\widehat{R_{n-1}P_{n-1}}, \widehat{P_{n-1}R_n}, \widehat{R_nR_{n-1}}\}$. We shall show in the next two lemmas that the region enclosed by the three arcs in $\mathcal{S}_1$ cannot be arbitrarily small. If all the arcs in H_n are straight lines, this fact is easy to prove. However, for circular-arc drawings, we need to take into account that the arcs can have different curvatures. Nonetheless, the requirement that the tangents of two incident arcs must form at least an angle $\alpha > 0$ will allow us to show that, regardless of the curvatures of the three arcs in $\mathcal{S}_1$, $Area(\mathcal{S}_1)$ is proportional to $|\overline{P_{n-1}Q_{n-1}}|^2$ as well as the area enclosed by $\widehat{P_{n-1}Q_{n-1}}$ and $\overline{P_{n-1}Q_{n-1}}$. Similarly, the areas of the regions enclosed by the arcs in $\mathcal{S}_2$ and $\mathcal{S}_3$ cannot be arbitrarily small.

Let Z_{n-1} be the midpoint of $\widehat{P_{n-1}Q_{n-1}}$. Consider the two circular arcs that pass through P_{n-1} and Z_{n-1} such that the tangents of the arcs form an angle α with $\widehat{P_{n-1}Q_{n-1}}$. Let $\widehat{a}$ be the arc that lies on the outside face of H_{n-1}. Let $\widehat{b}$ be the corresponding arc that passes through Q_{n-1} and Z_{n-1}, see Figure 3.

Lemma 4. *$Area(\mathcal{S}_1) \geq Area(\{\widehat{a}, \widehat{P_{n-1}Z_{n-1}}\})$.*

Proof: Let l be the perpendicular bisector of $\overline{P_{n-1}Q_{n-1}}$. Without loss of generality, assume that P_n lies on l or on the same side of l as Q_{n-1}. Notice that if $P_n \neq Z_{n-1}$, $\widehat{P_{n-1}P_n}$ is always above $\widehat{a}$ except at its endpoint, P_{n-1}. Otherwise, the angular resolution of Γ_n is violated or P_n lies below $\widehat{a}$ and hence on the wrong side of l. Furthermore, $\widehat{Q_{n-1}P_n}$ cannot intersect $\widehat{a}$, except possibly at Z_{n-1}. If it does, it crosses l and has to intersect $\widehat{P_{n-1}P_n}$ as well, contradicting the assumption that Γ_n is a planar drawing. Thus, both $\widehat{P_{n-1}P_n}$ and $\widehat{Q_{n-1}P_n}$

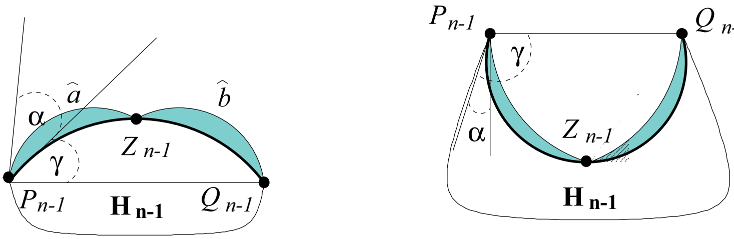


Fig. 3. Arcs $\widehat{a}$ and $\widehat{b}$ pass through P_{n-1}, Z_{n-1} and Q_{n-1}, Z_{n-1} respectively. Their tangents form an angle α with the tangents of $P_{n-1}\widehat{\ }Z_{n-1}$ and $Q_{n-1}\widehat{\ }Z_{n-1}$. The shape of the region bounded by $\widehat{a}$ and $P_{n-1}\widehat{\ }Z_{n-1}$ depends on the concavity/convexity of $P_{n-1}\widehat{\ }Q_{n-1}$ and α.

do not cross $\widehat{a}$; $\widehat{a}$ must lie in the region enclosed by $\mathcal{S}_1$. By symmetry, if P_n lies on the same side of l as P_{n-1}, then $\widehat{b}$ must lie in the region enclosed by $\mathcal{S}_1$. $\square$

Lemma 5. *There exists positive constants k_α and k'_α such that*

i. $Area(\{\widehat{a}, P_{n-1}\widehat{\ }Z_{n-1}\}) \geq k_\alpha |\overline{P_{n-1}Q_{n-1}}|^2$ *and*
ii. $Area(\{\widehat{a}, P_{n-1}\widehat{\ }Z_{n-1}\}) \geq k'_\alpha Area(\{P_{n-1}\widehat{\ }Q_{n-1}, \overline{P_{n-1}Q_{n-1}}\})$.

The proof of the above lemma is omitted from this extended abstract.

Note that k_α and k'_α are not dependent on γ and hence the result can be extended to the other arcs, $Q_{n-1}\widehat{\ }R_{n-1}$ and $R_{n-1}\widehat{\ }P_{n-1}$.

Theorem 2. *Any planar, circular-arc drawing of H_n that has constant angular resolution $\alpha > 0$ has area $\Omega(c_\alpha^n)$ where $c_\alpha > 1$.*

Proof: Let Γ_n^* be a planar, circular-arc drawing of H_n with minimum area A_n. Let B_{n-1} denote the area occupied by H_{n-1} in Γ_n^*. Clearly, $B_{n-1} \geq A_{n-1}$. Then,

$$
\begin{aligned}
A_n &\geq B_{n-1} + Area(\mathcal{S}_1) + Area(\mathcal{S}_2) + Area(\mathcal{S}_3) \\
&\geq B_{n-1} + \frac{1}{2}\Big[k_\alpha\,|\overline{P_{n-1}Q_{n-1}}|^2 + k'_\alpha\,Area(\{P_{n-1}\widehat{\ }Q_{n-1}, \overline{P_{n-1}Q_{n-1}}\}) + \\
&\quad k_\alpha\,|\overline{Q_{n-1}R_{n-1}}|^2 + k'_\alpha\,Area(\{Q_{n-1}\widehat{\ }R_{n-1}, \overline{Q_{n-1}R_{n-1}}\}) + \\
&\quad k_\alpha\,|\overline{R_{n-1}P_{n-1}}|^2 + k'_\alpha\,Area(\{R_{n-1}\widehat{\ }P_{n-1}, \overline{R_{n-1}P_{n-1}}\})\Big] \qquad (1)\\
&\geq B_{n-1} + \frac{\min(k_\alpha, k'_\alpha)}{2}\Big[|\overline{P_{n-1}Q_{n-1}}|^2 + |\overline{Q_{n-1}R_{n-1}}|^2 + |\overline{R_{n-1}P_{n-1}}|^2 \\
&\quad + Area(\{P_{n-1}\widehat{\ }Q_{n-1}, \overline{P_{n-1}Q_{n-1}}\}) + Area(\{Q_{n-1}\widehat{\ }R_{n-1}, \overline{Q_{n-1}R_{n-1}}\}) \\
&\quad + Area(\{R_{n-1}\widehat{\ }P_{n-1}, \overline{R_{n-1}P_{n-1}}\})\Big] \\
&\geq B_{n-1} + \frac{\min(k_\alpha, k'_\alpha)}{2}\Big[(Area(\{\overline{P_{n-1}Q_{n-1}}, \overline{Q_{n-1}R_{n-1}}, \overline{R_{n-1}P_{n-1}}\}) + \\
&\quad Area(\{P_{n-1}\widehat{\ }Q_{n-1}, \overline{P_{n-1}Q_{n-1}}\}) + Area(\{Q_{n-1}\widehat{\ }R_{n-1}, \overline{Q_{n-1}R_{n-1}}\}) + \\
&\quad Area(\{R_{n-1}\widehat{\ }P_{n-1}, \overline{R_{n-1}P_{n-1}}\})\Big] \qquad (2)\\
&\geq B_{n-1} + \frac{\min(k_\alpha, k'_\alpha)}{2} B_{n-1} \geq (1 + \frac{\min(k_\alpha, k'_\alpha)}{2}) A_{n-1}.
\end{aligned}
$$

Note that (1) follows from lemmas 4 and 5 and (2) from the fact that $B_{n-1} \geq Area(\{\overline{P_{n-1}Q_{n-1}}, \overline{Q_{n-1}R_{n-1}}, \overline{R_{n-1}P_{n-1}}\}) + Area(\{\widehat{P_{n-1}Q_{n-1}}, \overline{P_{n-1}Q_{n-1}}\}) + Area(\{\widehat{Q_{n-1}R_{n-1}}, \overline{Q_{n-1}R_{n-1}}\}) + Area(\{\widehat{R_{n-1}P_{n-1}}, \overline{R_{n-1}P_{n-1}}\})$. Let $c_\alpha = 1 + \min(k_\alpha, k'_\alpha)$. Since A_1 is at least some constant $a_1 > 0$, by induction, $A_n \geq c_\alpha^{n-1} a_1$. $\square$

Acknowledgements

We would like to thank Anhua Lin for helpful discussions and suggestions.

References

1. G. Di Battista, P. Eades, R. Tamassia, and I. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs.* Prentice Hall, Englewood Cliffs, NJ, 1999.
2. M. Chrobak and T. Payne. A linear-time algorithm for drawing planar graphs. *Inform. Process. Lett.*, 54:241–246, 1995.
3. H. de Fraysseix, J. Pach, and R. Pollack. How to draw a planar graph on a grid. *Combinatorica*, 10(1):41–51, 1990.
4. I. Fary. On straight lines representation of planar graphs. *Acta Sci. Math. Szeged*, 11:229–233, 1948.
5. M. Formann, T. Hagerup, J. Haralambides, M. Kaufmann, F. T. Leighton, A. Simvonis, Emo Welzl, and G. Woeginger. Drawing graphs in the plane with high resolution. *SIAM J. Comput.*, 22:1035–1052, 1993.
6. A. Garg and R. Tamassia. Planar drawings and angular resolution: Algorithms and bounds. In *Proc. 2nd Annu. European Sympos. Algorithms*, volume 855 of *Lecture Notes Comput. Sci.*, pages 12–23. Springer-Verlag, 1994.
7. M. T. Goodrich and C. G. Wagner. A framework for drawing planar graphs with curves and polylines. In *Graph Drawing '98*, pages 153–166, 1998.
8. C. Gutwenger and P. Mutzel. Planar polyline drawings with good angular resolution. In *Graph Drawing '98*, pages 167–182, 1998.
9. G. Kant. Drawing planar graphs using the *lmc*-ordering. In *Proc. 33th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 101–110, 1992.
10. G. Kant. Drawing planar graphs using the canonical ordering. *Algorithmica*, 16:4–32, 1996. (special issue on Graph Drawing, edited by G. Di Battista and R. Tamassia).
11. S. Malitz and A. Papakostas. On the angular resolution of planar graphs. *SIAM J. Discrete Math.*, 7:172–183, 1994.
12. W. Schnyder. Embedding planar graphs on the grid. In *Proc. 1st ACM-SIAM Sympos. Discrete Algorithms*, pages 138–148, 1990.
13. W. T. Tutte. How to draw a graph. *Proceedings London Mathematical Society*, 13(52):743–768, 1963.
14. K. Wagner. Bemerkungen zum vierfarbenproblem. *Jahresbericht der Deutschen Mathematiker-Vereinigung*, 46:26–32, 1936.

Drawing Graphs in the Hyperbolic Plane

Bojan Mohar*

Department of Mathematics, University of Ljubljana
1111 Ljubljana, Slovenia
bojan.mohar@uni-lj.si

Abstract. It is shown how one can draw graphs on surfaces of negative Euler characteristic by using hyperbolic geometry and hyperbolic circle packing representations. The same approach applies to drawings of hyperbolic tessellations.

1 Introduction

The purpose of this note is to offer a new approach for drawing graphs embedded in a surface Σ with negative Euler characteristic. Each such embedding can be changed by a homeomorphism to an embedding in a Riemann surface Σ_0 homeomorphic to Σ of constant negative curvature $\kappa < 0$ (say $\kappa = -1$) such that all edges of the graph become "straight line segments" (i.e., geodesic segments) on Σ_0. To get such a representation, we suggest using the *primal-dual circle packing algorithm* (abbreviated PDCP) which is presented in [9] (originally introduced by Lovász, see [3]) for the Euclidean plane and in [8] (see also [4], [5]) for the hyperbolic case treated in this paper. The PDCP algorithm determines a convex embedding in Σ_0 if (the universal cover of) the given graph G embedded in Σ is 3-connected. By considering the universal cover $\widetilde{\Sigma}_0$ of Σ_0, the problem of drawing G on Σ_0 transforms to the problem of drawing graphs in the *hyperbolic plane* $\mathbb{H}^2 \approx \widetilde{\Sigma}_0$ and then identifying the *fundamental domain* of Σ_0 in $\mathbb{H}^2$.

In Sections 2 and 3 we present two basic models of the hyperbolic plane — the upper half-plane and the unit disk model. Further on, we present the relation between the Euclidean and the hyperbolic geometry of the upper half-plane which enables us to develop the basic graph drawing primitives for drawings in $\mathbb{H}^2$. At the end we add a few examples.

It is to be hoped that our approach will stimulate further research related to drawings of graphs in the hyperbolic geometry. Our main motivation for this type of graph drawings are representations of graphs embedded in higher genus surfaces and drawings of hyperbolic tessellations. A related application to draw and navigate Internet sites and their connections using hyperbolic geometry was explored by Munzner [11].

For further reading on the hyperbolic geometry we refer the reader to [6], [12], [13], [14], [15].

* Supported in part by the Ministry of Science and Technology of Slovenia, Research Project J1-0502-0101-98.

J. Kratochvíl (Ed.): GD'99, LNCS 1731, pp. 127–136, 1999.

2 The Poincaré Half-Plane

The *Euclidean plane* $\mathbb{R}^2$ is endowed with the *Euclidean metric*

$$ds^2 = dx^2 + dy^2.$$

It is convenient to identify $\mathbb{R}^2$ with $\mathbb{C}$, the set of all complex numbers, so that the point $(x, y) \in \mathbb{R}^2$ is identified with $z = x + iy \in \mathbb{C}$.

The *Poincaré half-plane* $\mathbb{H}^2 = \{(x, y) \in \mathbb{R}^2 \mid y > 0\}$ is the upper half-plane endowed with the *hyperbolic metric*

$$ds^2 = \frac{dx^2 + dy^2}{y^2}.$$

If $\gamma(t) = x(t) + iy(t)$, $a \le t \le b$, is a C^1-curve in $\mathbb{H}^2$, then its *hyperbolic length* $\ell(\gamma)$ is defined as

$$\ell(\gamma) = \int_\gamma \frac{\sqrt{dx^2 + dy^2}}{y}. \tag{1}$$

The *geodesic lines* of $\mathbb{H}^2$ (also called *hyperbolic straight lines*) are either

(a) Euclidean semicircles with the center on the x-axis, or
(b) Euclidean straight lines in $\mathbb{H}^2$ which are perpendicular to the x-axis.

See Figure 1.

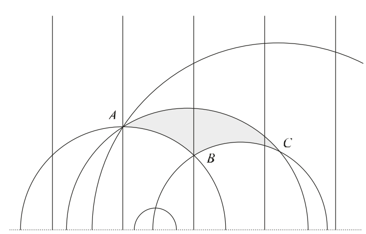


Fig. 1. Geodesic lines in $\mathbb{H}^2$

For any two points $z, w \in \mathbb{H}^2$, there is a unique hyperbolic straight line P containing them. The *hyperbolic distance* of z and w, denoted by $\mathrm{dist}_{\mathbb{H}}(z, w)$, is the hyperbolic length of the segment of P from z to w.

If A, B, C are points in $\mathbb{H}^2$, then the hyperbolic line segments joining A, B, and C determine the *hyperbolic triangle* $\Delta(ABC)$. In Figure 1, one of such triangles is displayed.

Theorem 1. *The rigid motions of* $\mathbb{H}^2$ *are precisely the transformations of the following forms:*

$$\text{(a)}\ f(z) = \frac{az + b}{cz + d} \qquad \textit{or} \qquad \text{(b)}\ f(z) = \frac{-a\bar{z} + b}{-c\bar{z} + d}$$

where a, b, c, d *are real numbers and* $ad - bc > 0$.

Let us mention two special cases. The *hyperbolic rotation* of $\mathbb{H}^2$ by the angle 2ϑ about the point $i = (0, 1)$ is the mapping

$$R_\vartheta(z) = \frac{\cos\vartheta\, z + \sin\vartheta}{-\sin\vartheta\, z + \cos\vartheta}. \tag{2}$$

The *translation* which maps $(a, b) \in \mathbb{H}^2$ to the point $(c, d) \in \mathbb{H}^2$ is the map

$$T_{a,b,c,d}(z) = \frac{dz + (bc - ad)}{b}. \tag{3}$$

Two geometric figures A, B in $\mathbb{H}^2$ (subsets of $\mathbb{H}^2$) are *congruent* if there is a rigid motion of $\mathbb{H}^2$ which transforms A into B.

Let $z = (x, y) \in \mathbb{H}^2$ and $r \in \mathbb{R}^+$. The (*hyperbolic*) *circle* of radius r centered at z is the set

$$C(z, r) = \{w \in \mathbb{H}^2 \mid \mathrm{dist}_{\mathbb{H}}(z, w) \leq r\}. \tag{4}$$

The *Euclidean circle* of radius r centered at z is defined as the set

$$C_E(z, r) = \{w \in \mathbb{C} : |z - w| \leq r\}. \tag{5}$$

Theorem 2. *Every hyperbolic circle is also a Euclidean circle. If $z = (x, y) \in \mathbb{H}^2$ and $r \in \mathbb{R}^+$, then $C(z, r) = C_E(z', r')$ where $z' = x + iy'$, $y' = y\ \mathrm{ch}(r)$, $r' = y\ \mathrm{sh}(r)$.*

Proof. We shall give only the proof of the second statement. Let $z_1 = (x, y_1)$ and $z_2 = (x, y_2)$ be the points in $\mathbb{H}^2$ at hyperbolic distance r from z which have the same first coordinate x as z and where $y_1 < y < y_2$. By (1) we easily get $y_1 = y \exp(-r)$ and $y_2 = y \exp(r)$. This implies that the Euclidean circle $C_E(z', r')$ has radius $r' = (y_2 - y_1)/2 = y\ \mathrm{sh}(r)$ and center $z' = (x, (y_1 + y_2)/2)$, so that $y' = y\ \mathrm{ch}(r)$.

If A, B, C are points in $\mathbb{H}^2$, then the hyperbolic line segments joining A, B, and C determine the *hyperbolic triangle* $\Delta(ABC)$. In Figure 1 one of such triangles is displayed. It is well known that the sum of the angles α, β, γ of a hyperbolic triangle is always less that π. It is also known that for any positive real numbers α, β, γ whose sum is $< \pi$ there is a hyperbolic triangle with angles α, β, γ. Moreover, any two such triangles are congruent and have the same hyperbolic area (which is equal to $\pi - \alpha - \beta - \gamma$).

Suppose that Δ is a hyperbolic triangle with angles α, β, γ and lengths of its sides equal to a, b, c (where the side of length a is opposite the angle α, etc.). These quantities are then related by the *Hyperbolic Law of Cosines*

$$\cos\alpha = \frac{\mathrm{ch}\, b \cdot \mathrm{ch}\, c - \mathrm{ch}\, a}{\mathrm{sh}\, b \cdot \mathrm{sh}\, c} \qquad \text{and} \qquad \mathrm{ch}\, a = \frac{\cos\beta \cdot \cos\gamma + \cos\alpha}{\sin\beta \cdot \sin\gamma} \tag{6}$$

and by the *Hyperbolic Law of Sines*

$$\frac{\sin\alpha}{\mathrm{sh}\, a} = \frac{\sin\beta}{\mathrm{sh}\, b} = \frac{\sin\gamma}{\mathrm{sh}\, c} \tag{7}$$

In the special case where $\gamma = \frac{\pi}{2}$ (the *right triangle*), we get

$$\mathrm{ch}\, c = \mathrm{ch}\, a \cdot \mathrm{ch}\, b = \mathrm{ctg}\,\alpha \cdot \mathrm{ctg}\,\beta. \tag{8}$$

3 The Unit Disk Model of the Hyperbolic Plane

The Möbius transformation

$$U(z) = \frac{iz + 1}{z + i} \tag{9}$$

maps $\mathbb{H}^2$ bijectively onto the (open) unit disk $\mathbb{D}^2$ in $\mathbb{C}$. The inverse mapping is

$$U^{-1}(w) = \frac{iw - 1}{-w + i} \tag{10}$$

The geodesics of $\mathbb{H}^2$ are mapped by $U(z)$ onto circular arcs with their endpoints on the boundary of $\mathbb{D}^2$ which meet the boundary of $\mathbb{D}^2$ perpendicularly, see Figure 2 below.

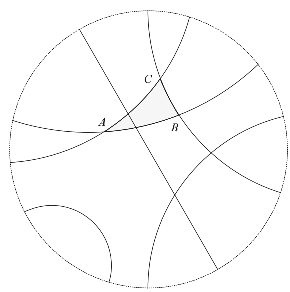


Fig. 2. Geodesic lines in the hyperbolic unit disk model

The mapping $U(z)$ determines the hyperbolic geometry on $\mathbb{D}^2$ which has the following Riemann metric

$$ds^2 = \frac{4(dx^2 + dy^2)}{(1 - x^2 - y^2)^2}. \tag{11}$$

If $\gamma(t) = x(t) + iy(t)$, $a \le t \le b$, is a C^1-curve in $\mathbb{D}^2$, then its *hyperbolic length* $\ell(\gamma)$ is defined as

$$\ell(\gamma) = \int_\gamma \frac{2\sqrt{dx^2 + dy^2}}{1 - x^2 - y^2}. \tag{12}$$

Theorem 3 below determines the rigid motions of $\mathbb{D}^2$:

Theorem 3. *The rigid motions of $\mathbb{D}^2$ are precisely the transformations of the following forms:*

$$\text{(a)}\ g(z) = \frac{az + \overline{c}}{cz + \overline{a}} \qquad \textit{or} \qquad \text{(b)}\ g(z) = \frac{-a\overline{z} + \overline{c}}{-c\overline{z} + \overline{a}}$$

where a and c are any complex numbers such that $|a| > |c|$.

Observe that $U(z)$ is a Möbius transformation. Since all Möbius transformations map circles (and halfspaces) to circles (and halfspaces), Theorem 2 implies:

Theorem 4. *Every hyperbolic circle in $\mathbb{D}^2$ of radius r and with center $w \in \mathbb{D}^2$ corresponding to the hyperbolic disk metric (11) is also a Euclidean circle $C_E(w', r')$, where $w' = \frac{-4\, e^r\, y}{-(1+e^r)^2+(-1+e^r)^2\, y^2}\, \frac{w}{|w|}$ and $r' = \frac{\left(-1+e^{2\,r}\right)\left(-1+y^2\right)}{-(1+e^r)^2+(-1+e^r)^2\, y^2}$.*

4 Hyperbolic Primal Dual Circle Packings

Let Σ be a surface. A *map* on Σ is a pair (G, Σ) where G is a connected graph that is 2-cell embedded in Σ. Given a map $M = (G, \Sigma)$, a *circle packing* of M is a set of (geodesic) circles in a Riemannian surface Σ_0 of constant curvature that is homeomorphic to Σ, one circle for each vertex of G, such that the following conditions are fulfilled:

(i) the interiors of circles are pairwise disjoint open disks,
(ii) for each edge $uv \in E(G)$, the circles corresponding to u and v touch, and
(iii) by putting a vertex v_D in the centre of each circle D and joining v_D by geodesics with all points on the boundary of D where the other circles touch D (or where D touches itself), we get a map on Σ_0 which is isomorphic to M.

Because of (iii) we also say to have a *circle packing representation* of M. The obtained map on Σ_0 is said to be a *straight-line* representation of M. Simultaneous circle packing representations of a map M and its dual map M^* are called a *primal-dual circle packing representation* of M if for any two edges

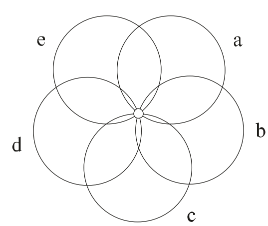


Fig. 3. The local rotation of G_0

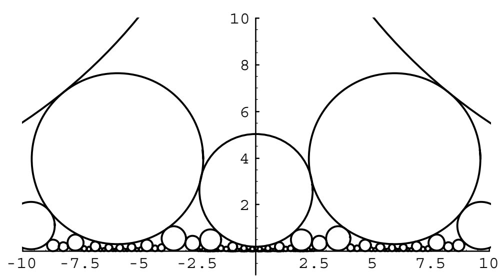


Fig. 4. The circle packing of G_0 in $\mathbb{H}^2$

$e = uv \in E(M)$ and $e^* = u^*v^* \in E(M^*)$ which are dual to each other, the circles C_u, C_v corresponding to e touch at the same point as the circles C_{u^*}, C_{v^*} of e^*, and C_u, C_{u^*} cross each other at that point perpendicularly. Having a primal-dual circle packing representation, each pair of dual edges intersects at the right angle. The obtained representations of the maps M and M^* on Σ_0 are easily seen to be *convex*, i.e., if x, y are points in the same face F of M (or M^*), then in F there is a geodesic (not necessarily a shortest one) joining x and y.

It was proved by Koebe [7], Andreev [1], [2], and Thurston [16] that if M is a triangulation, then it admits a circle packing representation (some of these results apply only for the 2-sphere). The proofs of Andreev and Thurston are existential (using a fixed point theorem) but Colin de Verdière [4], [5] found a constructive proof by means of a convergent process. Mohar [8] found a polynomial time algorithm that for a given map M (whose universal cover has 3-connected graph) and for a given $\varepsilon = 10^{-t}$ finds an ε-approximation for a circle packing of M into a surface of constant curvature (either +1, 0, or –1). The time used by that algorithm is polynomial in the size of the input $(= |V(M)| + |E(M)| + t)$.

5 Drawing Primitives

To draw graphs in the hyperbolic plane as suggested in the introduction we have to implement drawing primitives for drawing (hyperbolic) circles and hyperbolic line segments. We assume that we have a primitive for drawing Euclidean circles and Euclidean circular arcs:

`Circle(a,b,r)`: Draws the Euclidean circle with radius `r` centered at (`a`,`b`)$\in \mathbb{R}^2$.

`CircularArc(a,b,r,`α`,`β`)`: Draws the circular arc of `Circle(a,b,r)` corresponding to the angles from α to β.

`LineSegment(a,b,c,d)`: Draws the Euclidean straight line segment from (`a`,`b`) to (`c`,`d`).

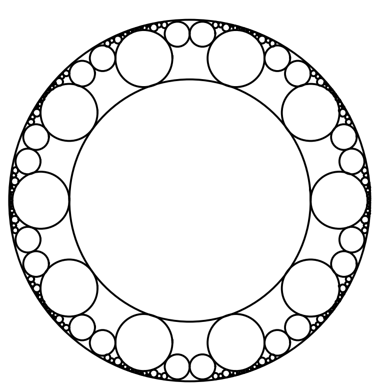


Fig. 5. The circle packing of G_0 in $\mathbb{D}^2$

We shall describe only the upper half-plane model. The unit disk model primitives are obtained by simply applying the Möbius transformation (9).

5.1 Circles

`HyperbolicCircle(x,y,r)`: Draws the hyperbolic circle with radius `r` centered at (`x,y`)$\in \mathbb{H}^2$. This primitive can be expressed as `Circle(x,y',r')` where `y'` and `r'` are given by Theorem 2.

5.2 Hyperbolic Line Segments

`HyperbolicLine(a,b,c,d)`: Draws the hyperbolic line segment from (`a,b`)$\in \mathbb{H}^2$ to (`c,d`)$\in \mathbb{H}^2$. If `a` $=$ `c`, this is just a vertical line segment. Otherwise, this is the circular arc of the Euclidean circle centered at $(x, 0)$ where $x = (c^2 + d^2 - a^2 - b^2)/(2(c - a))$.

`HyperbolicLineTangent(a,b,`α`,l)`: Draws the hyperbolic line segment from (`a,b`)$\in \mathbb{H}^2$ of length `l` which starts at (`a,b`) under the angle α with respect to the x-axis. This operation is easiest to implement by using hyperbolic rigid motions to determine the endpoint (`c,d`) of the segment (compose the translation $T_{a,b,0,1}$ defined by (3), the rotation R_φ by the angle 2φ, $\varphi = \frac{1}{2}(\frac{\pi}{2} - \alpha)$, about the point $i = (0, 1) \in \mathbb{H}^2$, determine the modified end of the line segment — the point on the y-axis above i at hyperbolic distance `l`, and map back). Finally, we just draw `HyperbolicLine(a,b,c,d)`. The same operation can be done directly by using `CircularArc(x,0,r,`φ_1`,`φ_2`)` where $x = a + b/\operatorname{tg}\alpha$, $r = \sqrt{(x-a)^2 + b^2}$, $\varphi_1 = \frac{\pi}{2} + \alpha$, and $\varphi_2 = 2\operatorname{arctg}(\operatorname{tg}(\frac{\pi}{4} + \frac{\alpha}{2})/\texttt{l})$. (Other formulae apply if $\alpha \in \{\frac{\pi}{2}, \frac{3\pi}{2}\}$.)

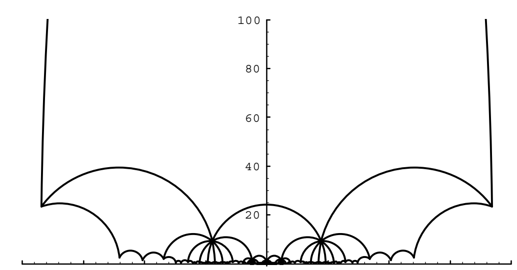


Fig. 6. The straight line drawing of G_0 in $\mathbb{H}^2$

6 Drawing in the Hyperbolic Plane

For the sake of simplicity we shall assume that G_0 is a graph embedded in some surface of negative Euler characteristic such that the graph of its universal cover

is 3-connected. (Otherwise we extend G_0 to a triangulation whose graph (and the graph of the universal cover) is necessarily 3-connected.)

Drawing algorithm consists of the following steps:

(1) Input the graph G_0 and the combinatorial description of its embedding Π_0 (see, e.g., [10]).
(2) Determine the Π_0-facial walks and construct the vertex-face incidence graph G (cf. [8]).
(3) Determine the radii r_v ($v \in V(G)$) of the primal-dual circle packing (cf. [8]).
(4) Start the BFS of the universal cover $\tilde{G}$ of G and draw its faces (respectively, the edges of G_0 or the circles of the circle packing of G_0) one after another until all the faces (respectively, edges or circles) have been drawn (at least) once. Identifying the first drawing of each of the faces gives the fundamental polygon of the surface.
(5) For a representation in the unit disk model, transform the drawing by applying the mapping $U(z)$ given by equation (9).

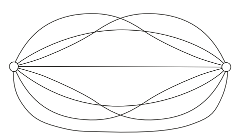


Fig. 7. The local rotation of G_1

7 Examples

Let G_0 be the graph with a single vertex v and with five loops a,b,c,d,e. Consider its embedding determined by the local rotation shown in Figure 3. This embedding has two facial walks, $A =$ adbec and $B =$ acebd. By Euler's formula, this is an embedding in the double torus. The vertex-face incidence graph has three vertices v, A, B, 5 edges joining v and A, and 5 edges joining v and B. The radii of the corresponding PDCP are $r_v = 1.61692$, $r_A = r_B = 1.06128$. Circle packings of G_0 in the hyperbolic upper half-plane and in the hyperbolic disk are shown in Figures 4 and 5, respectively. The corresponding straight line drawing in the hyperbolic plane is presented in Figure 6.

Figures 8 and 9 show the circle packing representations of the map G_1 shown in Figure 7.

Acknowledgment

The author is grateful to Arjana Žitnik for calculating the circle packing radii and for producing the actual circle packing drawings shown in Section 7.

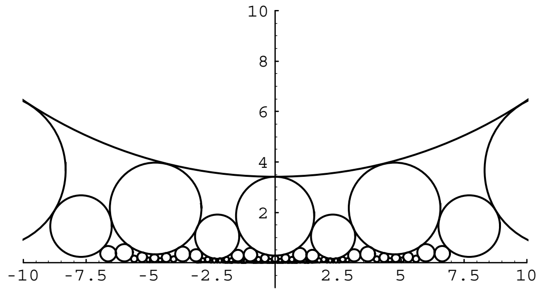


Fig. 8. The circle packing of G_1 in $\mathbb{H}^2$

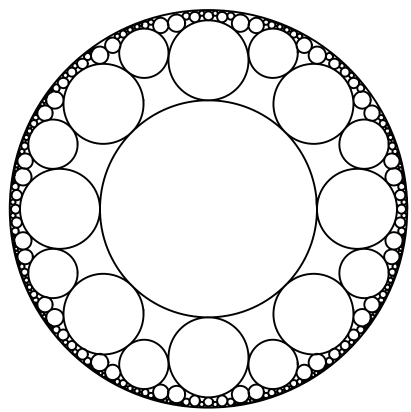


Fig. 9. The circle packing of G_1 in $\mathbb{D}^2$

References

1. E. M. Andreev, On convex polyhedra in Lobačevskiĭ spaces, Mat. Sb. (N. S.) 81 (1970) 445–478; Engl. transl. in Math. USSR Sb. 10 (1970) 413–440.
2. E. M. Andreev, On convex polyhedra of finite volume in Lobačevskiĭ space, Mat. Sb. (N. S.) 83 (1970) 256–260; Engl. transl. in Math. USSR Sb. 12 (1970) 255–259.
3. G. R. Brightwell, E. R. Scheinerman, Representations of planar graphs, SIAM J. Disc. Math. 6 (1993) 214–229.
4. Y. Colin de Verdière, Empilements de cercles: Convergence d'une méthode de point fixe, Forum Math. 1 (1989) 395–402.
5. Y. Colin de Verdière, Un principe variationnel pour les empilements des cercles, Invent. Math. 104 (1991) 655–669.
6. B. Iversen, Hyperbolic geometry, Cambridge Univ. Press, 1992.
7. P. Koebe, Kontaktprobleme auf der konformen Abbildung, Ber. Verh. Saechs. Akad. Wiss. Leipzig, Math.-Phys. Kl. 88 (1936) 141–164.
8. B. Mohar, Circle packings of maps in polynomial time, Europ. J. Combin. 18 (1997) 785–805.

9. B. Mohar, Circle packings of maps – The Euclidean case, Rend. Sem. Mat. Fis. (Milano), in press.
10. B. Mohar, C. Thomassen, Graphs on Surfaces, Johns Hopkins Univ. Press, to appear.
11. T. Munzner, Drawing large graphs with H3Viewer and Site Manager (system demonstration), in “Graph Drawing ’98”, Lecture Notes in Computer Science, Springer-Verlag, 1998, pp. 384–393.
12. A. Ramsay, R. D. Richtmeyer, Introduction to Hyperbolic Geometry, Springer-Verlag, New York, 1995.
13. J. G. Ratcliffe, Foundations of Hyperbolic Manifolds, Springer-Verlag, New York, 1994.
14. D. M. Y. Sommerville, The Elements of Non-Euclidean Geometry, Dover, New York, 1958.
15. S. Stahl, The Poincaré Half-Plane, Jones and Barlett Publishers, Boston, London, 1993.
16. W. P. Thurston, The geometry and topology of 3-manifolds, Princeton Univ. Lect. Notes, Princeton, NJ.

Graph Planarity and Related Topics

Robin Thomas

School of Mathematics, Georgia Institute of Technology
Atlanta, Georgia 30332-0160.
thomas@math.gatech.edu

Abstract. This compendium is the result of reformatting and minor editing of the author's transparencies for his talk delivered at the conference. The talk covered Euler's Formula, Kuratowski's Theorem, linear planarity tests, Schnyder's Theorem and drawing on the grid, the two paths problem, Pfaffian orientations, linkless embeddings, and the Four Color Theorem.

1 Basics

A *plane* graph is a graph drawn in the plane with no crossings. A graph is *planar* if it can be drawn in the plane with no crossings. Edges can be represented by homeomorphic images of $[0,1]$, continuous images of $[0,1]$, piecewise-linear images of $[0,1]$, or, by Fáry's theorem [11], by straight-line segments, and the class of planar graphs remains the same.

Euler'S Formula. For a connected plane graph G, $|V(G)|+|F(G)| = |E(G)|+2$.

Corollary. If a planar graph G has at least three vertices, then $|E(G)| \leq 3|V(G)| - 6$. If, in addition, G has no triangles, then $|E(G)| \leq 2|V(G)| - 4$.

Kuratowski's Theorem. A graph is planar if and only if it has no subgraph isomorphic to a subdivision of K_5 or $K_{3,3}$.

The "only if" part follows easily from the fact that K_5 and $K_{3,3}$ have too many edges. The "if" part is much harder. An elegant proof was found by Thomassen [40].

2 Testing Planarity in Linear Time

There are two classical linear-time planarity tests by Hopcroft and Tarjan [13], and by Lempel, Even and Cederbaum [19] and Booth and Lueker [7]. I will outline a new algorithm, found independently by Shih and Hsu [36] and Boyer and Myrvold [8]. A self-contained description of the algorithm may be found in [37].

Outline. Find a DFS tree T, and number its vertices $v_1, v_2, \ldots, v_n$ so that the numbers are decreasing along every path of T from the root, and the graph induced by the vertices $v_i, v_{i+1}, \ldots, v_n$ is connected for all $i = 1, 2, \ldots, n$. The

J. Kratochvíl (Ed.): GD'99, LNCS 1731, pp. 137–144, 1999.

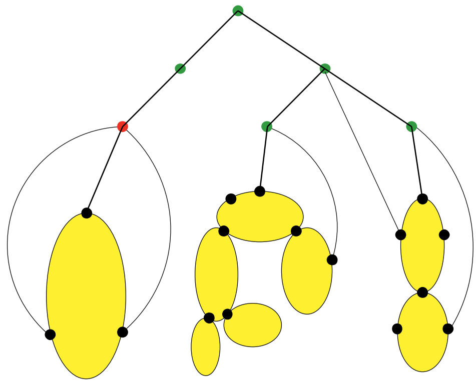


Fig. 1.

algorithm recursively adds the vertices $v_1, v_2, \ldots, v_n$ in the order listed. At the beginning of the ith iteration we have a certain "tree-structure" illustrated in Figure 1.

The shaded ovals represent blocks of the graph induced by $v_1, v_2, \ldots, v_{i-1}$. Each of these blocks B is embedded in the plane, and all edges joining it to the rest of the graph have one end incident with the infinite region of B. We now add the vertex v_i to this structure; it can be shown that if this cannot be done, then the graph has a K_5 or $K_{3,3}$ subdivision.

3 Schnyder's Theorem and Drawing on a Grid

Theorem. (Schnyder [31]) A graph is planar if and only if its vertex-edge poset has dimension at most three.

The latter is equivalent to the existence of linear orderings $\leq_1, \leq_2, \leq_3$ satisfying:
 $(*)$ $\forall uv \in E(G)\ \forall w \in V(G) - \{u,v\}\ \exists i$ such that $u \leq_i w$ and $v \leq_i w$.

Given $\leq_1, \leq_2, \leq_3$ there exists a 1-1 map $v \in V(G) \to (v_1, v_2, v_3) \in \mathbf{R}^3$ such that
 (i) $v_1 + v_2 + v_3 = 2n - 5$ for all $v \in V(G)$
 (ii) $v_i \in [0, 2n-5]$ is an integer
 (iii) for $uv \in E(G)$ we have $u \leq_i v$ if and only if $u_i \leq v_i$

and the coordinatewise orderings satisfy $(*)$. It follows that this gives a straight-line embedding on a grid. The latter was independently obtained by de Fraysseix, Pach and Pollack [12], and since then has been the subject of extensive research.

4 Colin de Verdiere's Parameter

Let $\mu(G)$ be the maximum corank of a matrix M satisfying
 (i) for $i \neq j$, $M_{ij} = 0$ if $ij \notin E$ and $M_{ij} < 0$ otherwise,
 (ii) M has exactly one negative eigenvalue,

(iii) if X is a symmetric $n \times n$ matrix such that $MX = 0$ and $X_{ij} = 0$ whenever $i = j$ or $ij \in E$, then $X = 0$.

Theorem. (Colin de Verdière [10]) $\mu(G) \leq 3$ if and only if G is planar.

5 Separators

Let G be a graph on n vertices. A *separator* in G is a set S such that every component of $G \backslash S$ has at most $2n/3$ vertices.

Theorem. (Lipton, Tarjan [20]) Every planar graph has a separator of size at most $\sqrt{8n}$.

Alon, Seymour and Thomas [2] found a simpler proof and improved the constant to $\sqrt{4.5n}$. In [1] they proved that graphs not contractible to K_t have a separator of size at most $\sqrt{t^3 n}$.

6 The Two Paths Problem

Given a graph G and $s_1, s_2, t_1, t_2 \in V(G)$ do there exist disjoint paths P_1, P_2 in G such that P_i has ends s_i and t_i?

A reduction. Let $V(G) = A \cup B$, where $|A \cap B| \leq 3$, no edge has one end in $A - B$ and the other in $B - A$, $s_1, s_2, t_1, t_2 \in A$, and, subject to that, $|A \cap B|$ is minimum. Let H be obtained from G by deleting $B - A$ and adding an edge joining every pair of vertices in $A \cap B$. Then the paths exist in G if and only if they exist in H. We say that G is *reduced* if $|V(G)|$ cannot be lowered by performing this operation.

Theorem. (Seymour [33], Shiloach [35], Thomassen [39]) If a graph G is reduced, then the paths exist if and only if G cannot be drawn in a disk with s_1, s_2, t_1, t_2 drawn on the boundary of the disk in order.

7 Pfaffian Orientations

An orientation of a graph G is *Pfaffian* if every even cycle C such that $G \backslash V(C)$ has a perfect matching has an odd number of edges directed in either direction of the cycle. If a graph has a Pfaffian orientation, then the number of perfect matchings can be computed in polynomial time.

Theorem. (Kasteleyn [16],[17]) Every planar graph has a Pfaffian orientation.

The decision problem whether a bipartite graph has a Pfaffian orientation is equivalent to:

- Pólya's permanent problem [24]
- the even directed cycle problem [34], [41], [43], [44]
- a hypergraph problem [32]
- the sign nonsingular matrix problem [9], [18], [42].

Theorem. (McCuaig [23], Robertson, Seymour and Thomas [29]) A bipartite graph has a Pfaffian orientation if and only if it can be obtained from planar bipartite graphs and the Heawood graph by means of 0-, 2- and 4-sums.

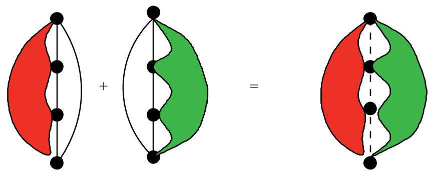


Fig. 2. 4-sum.

Corollary. There exists a cubic algorithm to solve the above-mentioned problems.

8 Linkless Embeddings

A (piecewise linear) embedding of a graph in $\mathbf{R}^3$ is *linkless* if every two disjoint cycles have zero linking number.

Theorem. (Robertson, Seymour and Thomas [28]) For a graph G the following conditions are equivalent:
(i) G has a linkless embedding
(ii) G has no subgraph that can be contracted onto one of seven specific graphs (K_6, $K_{3,3}$+vrtx, $K_{4,4}\backslash e$, ..., Petersen)
(iii) G has a flat embedding

An embedding of a graph G in $\mathbf{R}^3$ is *flat* if every cycle bounds a disk disjoint from the rest of G.

Remark. If G is planar, then an embedding $G \hookrightarrow \mathbf{R}^3$ is flat if and only if the image of G is a subset of an embedded sphere.

Theorem. (Robertson, Seymour and Thomas [28]) An embedding of a graph G in $\mathbf{R}^3$ is flat if and only if the fundamental group of the complement of every subgraph is free.

Theorem. (Robertson, Seymour and Thomas [28]) If a 4-connected graph has a flat embedding, then it has a unique flat embedding. More generally, any two flat embeddings of the same graph are related by "3-switches".

Theorem. (Lovász and Schrijver [21]) A graph G has a flat embedding if and only if $\mu(G) \leq 4$.

An embedding of a graph G in $\mathbf{R}^3$ is *knotless* if every cycle of G bounds a disk. No characterization of knotless graphs is known, but the following is a consequence of the Robertson-Seymour theory [26], [27].

Theorem. There exists a cubic algorithm to test whether an input graph has a knotless embedding.

The above only guarantees the existence of an algorithm. In fact, at present no explicit algorithm (let alone a polynomial-time one) to decide whether a graph has a knotless embedding is known.

9 A Digression

The cross-product is not associative, and so

$$v_1 \times v_2 \times \cdots \times v_n \tag{1}$$

is not well-defined if $n > 2$. If we put in enough brackets to make it well-defined, we get a *bracketing*. For example, $((v_1 \times v_2) \times (v_3 \times v_4)) \times v_5$ is a bracketing.

Theorem. (Kauffman [14]) For every two bracketings of (1) there exists an assignment of the standard unit vectors in $\mathbf{R}^3$ to $v_1, \ldots, v_n$ such that the evaluations of the two bracketings are equal and nonzero.

Theorem. (Kauffman [14]) The above theorem is equivalent to the Four-Color Theorem.

10 The Four-Color Theorem

Theorem. Every planar graph is 4-colorable.

Comments:
- Simple statement, yet proof is long.
- A 1976 proof by Appel and Haken [3], [4], reprinted in [5].
- A simpler proof by Robertson, Sanders, Seymour and Thomas [25].
- Both proofs are computer assisted.
- There are over two dozen equivalent formulations (in terms of graphs [30], vector cross products [14], Lie algebras [6], divisibility [22], Temperley-Lieb algebras [15])
- There are interesting conjectured generalizations.
- See [38] for a survey.

11 Outline of a Proof of the 4CT

Let G be a counterexample with $|V(G)|$ minimum. It can be shown that G is an internally 6-connected triangulation, where a graph G is *internally* 6*-connected* if it is 5-connected, and for every vertex cut X of size five, $G\backslash X$ has exactly two components, one of which has only one vertex.

A *configuration* is a pair consisting of a near-triangulation K (i.e., a planar graph such that every region except possibly the infinite one is bounded by a cycle of length three) and a labeling of the vertices of K by integers. A configuration K *appears* in a triangulation T if K is an induced subgraph of T and for every vertex of K its label equals its degree in T. We exhibit a set $\mathcal{U}$ of 633 configurations such that the following two theorems hold. Some configurations in this set are depicted in Figure 3, where the labeling is indicated by vertex shapes—a solid circle stands for 5, a dot (or what appears as no symbol at all) for 6, a hollow circle for 7 and a hollow square for 8.

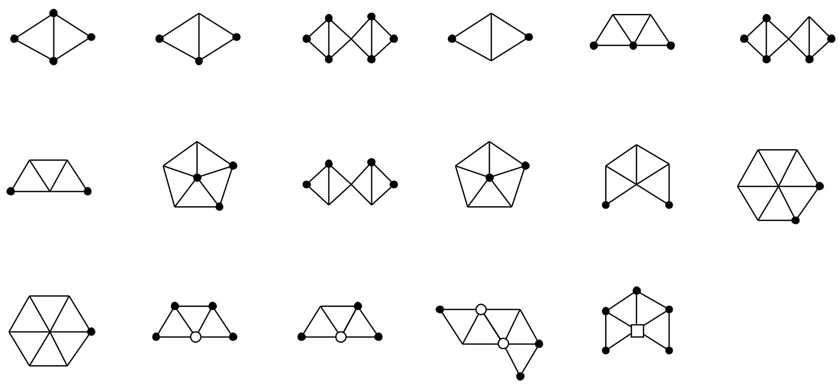


Fig. 3. Examples of configurations.

Theorem 1. No member of $\mathcal{U}$ appears in a minimal counterexample to the 4CT.

Theorem 2. For every internally 6-connected triangulation T, some member of $\mathcal{U}$ appears in T.

Thus Theorems 1 and 2, together with the fact that every minimal counterexample to the 4CT is an internally 6-connected triangulation, imply the Four-Color Theorem. The proof of Theorem 2 proceeds as follows. In a triangulation T with n vertices and e edges we have $e = 3n - 6$ and so

$$d_1 + d_2 + \cdots + d_n = 2e = 6n - 12,$$

where $d_1, d_2, \ldots , d_n$ are the degrees of vertices of T. This can be rewritten as

$$(6 - d_1) + (6 - d_2) + \cdots + (6 - d_n) = 12.$$

Initially, a vertex of degree d will receive a *charge* of $6 - d$. Thus the sum of the charges is 12. Charges will be redistributed according to certain rules, but the

sum will remain the same. Thus there is a vertex v of positive charge. We show that a reducible configuration appears in the second neighborhood of v.

Corollary. There is a quadratic algorithm to 4-color planar graphs.

References

1. N. Alon, P. D. Seymour and R. Thomas, A separator theorem for non-planar graphs, *J. Amer. Math. Soc.* **3** (1990), 801–808.
2. N. Alon, P. D. Seymour and R. Thomas, Planar separators, *SIAM J. Disc. Math.* **7** (1994), 184–193.
3. K. Appel and W. Haken, Every planar map is four colorable, Part I: discharging, *Illinois J. of Math.* **21** (1977), 429–490.
4. K. Appel, W. Haken and J. Koch, Every planar map is four colorable, Part II: reducibility, *Illinois J. of Math.* **21** (1977), 491–567.
5. K. Appel and W. Haken, Every planar map is four colorable, *Contemp. Math.* **98** (1989).
6. D. Bar-Natan, Lie algebras and the four color theorem, *Combinatorica* **17** (1997), 43–52.
7. K. S. Booth and G. S. Luecker, Testing the consecutive ones property, interval graphs and graph planarity using PQ-tree algorithms, *J. Comput. Syst. Sci.* **13** (1976), 335–379.
8. J. Boyer and W. Myrvold, Stop minding your P's and Q's: A simplified $O(n)$ planar embedding algorithm, *Tenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)* (1999), 140–146.
9. R. A. Brualdi and B. L. Shader, Matrices of sign-solvable linear systems, *Cambridge Tracts in Mathematics* **116** (1995).
10. Y. Colin de Verdière, Sur un nouvel invariant des graphes et un critère de planarité, *J. Combin. Theory Ser. B* **50**, (1990), 11–21.
11. I. Fáry, On straight line representations of planar graphs, *Acta Univ. Szeged. Sect. Sci. Math.* **11** (1948), 229-233.
12. H. de Fraysseix, J. Pach and R. Pollack, How to draw a graph on a grid, *Combinatorica* **10** (1990), 41–51.
13. J. E. Hopcroft and R. E. Tarjan, Efficient planarity testing, *J. Assoc. Comput. Mach.* **21** (1974), 549–568.
14. L. H. Kauffman, Map coloring and the vector cross product, *J. Combin. Theory Ser. B* **48** (1990), 145–154.
15. L. H. Kauffman and R. Thomas, Temperley-Lieb algebras and the Four-Color Theorem, manuscript.
16. P. W. Kasteleyn, Dimer statistics and phase transitions, *J. Math. Phys.* **4** (1963), 287–293.
17. P. W. Kasteleyn, Graph theory and crystal physics, in Graph Theory and Theoretical Physics (ed. F. Harary), Academic Press, New York, 1967, 43–110.
18. V. Klee, R. Ladner and R. Manber, Sign-solvability revisited, *Linear Algebra Appl.* **59** (1984), 131–158.
19. A. Lempel, S. Even and I. Cederbaum, An algorithm for planarity testing of graphs, In: Theory of Graphs, P. Rosenstiehl ed., Gordon and Breach, New York, 1967, 215–232.
20. R. J. Lipton and R. E. Tarjan, A separator theorem for planar graphs, *SIAM J. Appl. Math.* 36 (1979), 177-189.

21. L. Lovász and A. Schrijver, A Borsuk theorem for antipodal links and a spectral characterization of linklessly embeddable graphs, *Proc. Amer. Math. Soc.* **126** (1998), 1275–1285.
22. Y. Matiyasevich, The Four Colour Theorem as a possible corollary of binomial summation, manuscript.
23. W. McCuaig, Pólya's permanent problem, manuscript (81 pages), June 1997.
24. G. Pólya, Aufgabe 424, *Arch. Math. Phys. Ser.* **20** (1913), 271.
25. N. Robertson, D. P. Sanders, P. D. Seymour and R. Thomas, The four-colour theorem, *J. Combin. Theory Ser. B* **70** (1997), 2-44.
26. N. Robertson and P. D. Seymour, Graph Minors XIII. The disjoint paths problem, *J. Combin. Theory Ser. B* **63** (1995), 65–110.
27. N. Robertson and P. D. Seymour, Graph Minors XX. Wagner's conjecture, manuscript.
28. N. Robertson, P. D. Seymour and R. Thomas, Sach's linkless embedding conjecture, *J. Combin. Theory Ser. B* **64** (1995), 185–227.
29. N. Robertson, P. D. Seymour and R. Thomas, Permanents, Pfaffian orientations, and even directed circuits, manuscript.
30. T. L. Saaty, Thirteen colorful variations on Guthrie's four-color conjecture, *Am. Math. Monthly* **79** (1972), 2–43.
31. W. Schnyder, Planar graphs and poset dimension, *Order* **5** (1991), 323-343.
32. P. D. Seymour, On the two-colouring of hypergraphs, *Quart. J. Math. Oxford* **25** (1974), 303–312.
33. P. D. Seymour, Disjoint paths in graphs, *Discrete Math.* **29** (1980), 293-309.
34. P. D. Seymour and C. Thomassen, Characterization of even directed graphs, *J. Combin. Theory Ser. B* **42** (1987), 36-45.
35. Y. Shiloach, A polynomial solution to the undirected two paths problem, *J. Assoc. Comp. Machinery*, **27**, (1980), 445-456.
36. W.-K. Shih and W.-L. Hsu, A new planarity test, *Theoret. Comp. Sci.* **223** (1999), 179–191.
37. R. Thomas, Planarity in linear time, unpublished class notes, available from `http://www.math.gatech.edu/~thomas/planarity.ps`.
38. R. Thomas, An update on the Four-Color Theorem, *Notices Amer. Math. Soc.* **45** (1998), 848–859.
39. C. Thomassen, 2-linked graphs, *Europ. J. Combinatorics* **1** (1980), 371–378.
40. C. Thomassen, Kuratowski's theorem, *J. Graph Theory* **5** (1981), 225–241.
41. C. Thomassen, Even cycles in directed graphs, *European J. Combin.* **6** (1985), 85–89.
42. C. Thomassen, Sign-nonsingular matrices and even cycles in directed graphs, *Linear Algebra and Appl.* **75** (1986), 27–41.
43. C. Thomassen, The even cycle problem for directed graphs, *J. Amer. Math. Soc.* **5** (1992), 217–229.
44. V. V. Vazirani and M. Yannakakis, Pfaffian orientations, 0-1 permanents, and even cycles in directed graphs, *Discrete Appl. Math.* **25** (1989), 179–190.

Grid Drawings of Four-Connected Plane Graphs

Kazuyuki Miura[1], Shin-ichi Nakano[2], and Takao Nishizeki[1]

[1] Graduate School of Information Sciences
Tohoku University, Aoba-yama 05, Sendai 980-8579, Japan
miura@nishizeki.ecei.tohoku.ac.jp
nishi@ecei.tohoku.ac.jp
[2] Department of Computer Science, Faculty of Engineering, Gunma University;
Mailing address: 1-5-1 Tenjin-cho, Kiryu, Gunma, 376-8515 Japan
nakano@cs.gunma-u.ac.jp

Abstract. A grid drawing of a plane graph G is a drawing of G on the plane so that all vertices of G are put on plane grid points and all edges are drawn as straight line segments between their endpoints without any edge-intersection. In this paper we give a very simple algorithm to find a grid drawing of any given 4-connected plane graph G with four or more vertices on the outer face. The algorithm takes time $O(n)$ and needs a rectangular grid of width $\lceil n/2 \rceil - 1$ and height $\lceil n/2 \rceil$ if G has n vertices. The algorithm is best possible in the sense that there are an infinite number of 4-connected plane graphs any grid drawings of which need rectangular grids of width $\lceil n/2 \rceil - 1$ and height $\lceil n/2 \rceil$.

1 Introduction

Recently automatic aesthetic drawing of graphs has created intense interest due to their broad applications, and as a consequence, a number of drawing methods have come out [1,2,3,4,5,6,7,8,9,10,11,13,15]. The most typical method is *the straight line drawing* in which all edges of a graph are drawn as straight line segments without any edge-intersection. Every plane graph has a straight line drawing [8,14,16]. However, not every straight line drawing is an aesthetic drawing since many vertices may be concentrated in a small area.

A straight line drawing of a plane graph G is called a *grid drawing* of G if the vertices of G are put on grid points of integer coordinates. Of course, the distance between any two vertices in the drawing is at least 1. *The integer grid* of size $W \times H$ consists of $W+1$ vertical segments and $H+1$ horizontal segments, and has a rectangular contour. W and H are called the *width* and *height* of the integer grid, respectively. It is known that every plane graph of $n \geq 3$ vertices has a grid drawing on an $(n-2) \times (n-2)$ grid, and that such a grid drawing can be found in linear time [3,6,9,13]. It is also shown that, for each $n \geq 3$, there exists a plane graph which needs a grid of size at least $\lfloor 2(n-1)/3 \rfloor \times \lfloor 2(n-1)/3 \rfloor$ for any grid drawing [4,9]. It has been conjectured that every plane graph has a grid drawing on a $\lceil 2n/3 \rceil \times \lceil 2n/3 \rceil$ grid, but it is still an open problem. On the other hand, a restricted class of graphs has a more compact grid drawing. For

J. Kratochvíl (Ed.): GD'99, LNCS 1731, pp. 145–154, 1999.

example, if G is a 4-connected plane graph and has at least four vertices on its outer face, then G has a grid drawing on a $W \times H$ grid such that $W + H \leq n$, $W \leq (n+3)/2$ and $H \leq 2(n-1)/3$, and one can find such a grid drawing in linear time [10]. However, the algorithm is rather complicated.

In this paper, we give a very simple algorithm which finds a grid drawing of any given 4-connected plane graph G on a $W \times H$ grid such that $W = \lceil n/2 \rceil - 1$ and $H = \lceil n/2 \rceil$ in linear time if G has four or more vertices on the outer face. Since $W = \lceil n/2 \rceil - 1$ and $H = \lceil n/2 \rceil$, $W + H \leq n$. Thus our bounds on W and H are better than He's bounds [10]. Our bounds are indeed best possible, because there exist an infinite number of 4-connected plane graphs, for example the nested quadrangles depicted in Fig. 1, which need grids of size at least $W = \lceil n/2 \rceil - 1$ and $H = \lceil n/2 \rceil$ for any grid drawing. An aspect ratio of a drawing obtained by the algorithm [10] may be $1 : 4/3$, while the ratio of our algorithm is always $1 : 1$. Both our algorithm and the proof of its correctness are very simple, and it is quite easy to understand them.

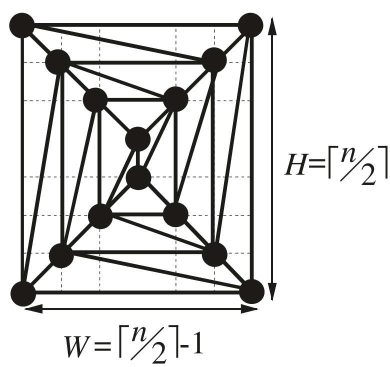


Fig. 1. Nested quadrangles attaining our bounds.

The outline of our algorithm is as follows. One can assume without loss of generality that a given graph G is internally triangulated as illustrated in Fig. 2(a). First, we find a "4-canonical ordering" of G [12]. Using the ordering, we then divide G into two graphs G' and G'', each of which has about $n/2$ vertices as illustrated in Fig. 2(b) where G' and G'' are shaded. Next, we draw the plane subgraph G' in an isosceles right-angled triangle whose base has length $W' = n/2 - 1$ and whose height is $H' = W'/2$, as illustrated in Fig. 2(c). Similarly, we draw G'' in the same triangle with its upside down. In Fig. 2(c) the two triangles are drawn by thick dotted lines. We place the two triangles so that their vertices opposite to their bases are separated by distance 1. Finally, we combine the drawings of G' and G'' to obtain a grid drawing of G, as illustrated in Fig. 2(d). The drawing of G has sizes $W = W' = n/2 - 1$ and $H = 2H' + 1 = W' + 1 = n/2$.

The remainder of the paper is organized as follows. In Section 2, we give some definitions and lemmas, and present our algorithm and a main theorem. In Section 3, we show how to draw $G^{'}$ and $G^{''}$.

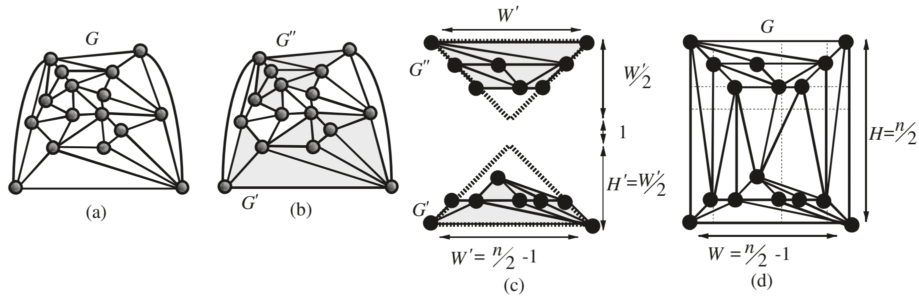


Fig. 2. Drawing process of our algorithm.

2 Main Theorem

In this section we first introduce some definitions and lemmas, and then present our algorithm and a main theorem.

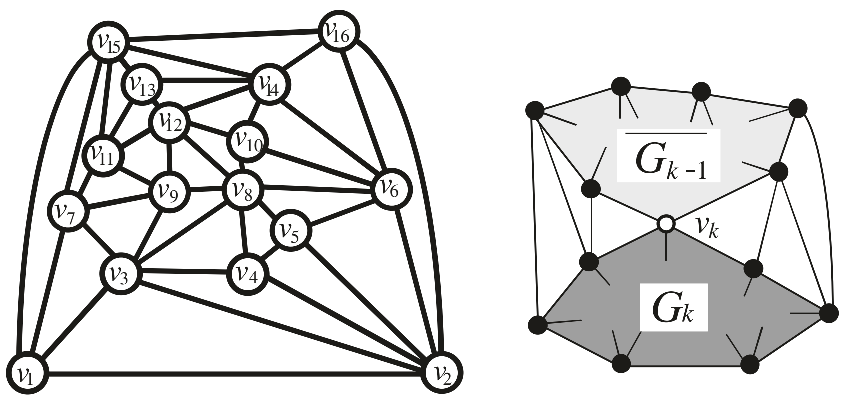
Let $G = (V, E)$ be a simple connected graph having no multiple edge or loop. V is the vertex set and E is the edge set of G. Let n be the number of vertices of G. An edge joining vertices u and v is denoted by (u, v). The *degree* of a vertex v in G is the number of neighbors of v in G, and is denoted by $d(v, G)$. The *connectivity* $\kappa(G)$ of a graph G is the minimum number of vertices whose removal results in a disconnected graph or a single-vertex graph K_1. A graph G is *k-connected* if $\kappa(G) \geq k$. Let $x(v)$ and $y(v)$ be the x- and y-coordinates of vertex $v \in V$, respectively.

A graph is *planar* if it can be embedded in the plane so that no two edges intersect geometrically except at a vertex to which they are both incident. A *plane graph* is a planar graph with a fixed embedding. A plane graph divides the plane into connected regions called *faces*. We denote the boundary of a face by a clockwise sequence formed by the vertices and edges on the boundary. We call the boundary of the outer face of a plane graph G *the contour* of G, and denote it by $C_o(G)$. A plane graph G is *internally triangulated* if all inner faces of G are triangles. We can assume without loss of generality that a given graph G is internally triangulated. Otherwise, we internally triangulate G by adding some new edges to G, find a drawing of the resulting graph, and finally remove the added edges to obtain a drawing of G.

We then give a definition of a 4-canonical ordering of a plane graph G [12], on which both our algorithm and He's [10] are based. The 4-canonical ordering is a generalization of the "canonical ordering" [9] which is used to find a grid drawing

of triangulated plane graph. Let $\Pi = (v_1, v_2, \cdots, v_n)$ be an ordering of set V. Fig. 3(a) illustrates an ordering of the graph G in Fig. 2(a). Let G_k, $1 \leq k \leq n$, be the plane subgraph of G induced by the vertices in $\{v_1, v_2, \cdots, v_k\}$, and let $\overline{G_k}$ be the plane subgraph of G induced by the vertices in $\{v_{k+1}, v_{k+2}, \cdots, v_n\}$. Thus $G = G_n = \overline{G_0}$. In Fig. 3(b), G_k is darkly shaded, while $\overline{G_{k-1}}$ is lightly shaded. We say that Π is a *4-canonical ordering* of G if the following two conditions are satisfied:

(co1) v_1 and v_2 are the ends of an edge on $C_o(G)$, and v_n and v_{n-1} are the ends of an edge on $C_o(G)$; and
(co2) for each k, $3 \leq k \leq n-2$, v_k is on $C_o(G_k)$, $d(v_k, G_k) \geq 2$, and $d(v_k, \overline{G_{k-1}}) \geq 2$.



(a) (b)

Fig. 3. (a) A 4-canonical ordering of a 4-connected plane graph of $n = 16$ vertices, and (b) an illustration for the condition (co2).

Although the definition of a 4-canonical ordering above is slightly different from that in [12], they are effectively equivalent each other. The following lemma is known.

Lemma 1. [12] Let G be a 4-connected plane graph having at least four vertices on $C_o(G)$. Then G has a 4-canonical ordering Π, and Π can be found in linear time.

We are now ready to present our algorithm *Draw*.

Procedure Draw(G)
begin
Find a 4-canonical ordering $\Pi = (v_1, v_2, \cdots, v_n)$ of a given 4-connected plane graph $G = (V, E)$;

2 Divide G into two subgraphs G' and G'' where $n' = \lceil n/2 \rceil$, $G' = G_{n'}$, and $G'' = \overline{G_{n'}}$;
3 Draw G' in an isosceles right-angled triangle whose base has length $W' = \lceil n/2 \rceil - 1$ and whose height is $H' = W'/2$;
4 Draw G'' in the same triangle with its upside down;
5 Place the two triangles so that their vertices opposite to their bases are separated by distance 1 and have the same x-coordinate;
6 Draw every edge of G joining a vertex in G' and a vertex in G'' by a straight line segment;
end.

We say that a curve in the plane is *x-monotone* if the intersection of the curve and any vertical line is a single point when it is nonempty. We then have the following lemma for the drawing of G', the proof of which will be given later in Section 3.

Lemma 2. One can find in linear time a grid drawing of G' satisfying the following conditions (a), (b) and (c):
(a) the drawing is in an isosceles right-angled triangle whose base has length $W' = \lceil n/2 \rceil - 1$ and whose height is $H' = W'/2$, and edge (v_1, v_2) is drawn as the base of the triangle;
(b) the absolute value of the slope of every edge on $C_o(G')$ is at most 1; and
(c) the drawing of the path going clockwise on $C_o(G')$ from v_1 to v_2 is x-monotone.

If $\Pi = (v_1, v_2, \cdots, v_n)$ is a 4-canonical ordering, then the reversed ordering $\Pi' = (v_n, v_{n-1}, \cdots, v_1)$ is also a 4-canonical ordering. Therefore G'' has a grid drawing in the same triangle. Hence we have the following main theorem.

Theorem 1. Algorithm *Draw* finds in linear time a grid drawing of a given 4-connected plane graph G on a $W \times H$ grid such that $W = \lceil n/2 \rceil - 1$ and $H = W + 1 = \lceil n/2 \rceil$ if there are four or more vertices on the contour $C_o(G)$.

Proof. If step 6 in *Draw* does not introduce any edge-intersection, then algorithm *Draw* correctly finds a grid drawing of G and clearly the size of a drawing of G satisfies $W = W'$ and $H = H' + H' + 1$. (See Fig. 2.) By Lemma 2(a) $W' = \lceil n/2 \rceil - 1$ and $H' = W'/2$. Therefore $W = \lceil n/2 \rceil - 1$ and $H = W + 1 = \lceil n/2 \rceil$. Thus we shall show that step 6 does not introduce any edge-intersection.

An oblique side of each isosceles right-angled triangle has slope $+1$ and the other oblique side has slope -1. The two vertices of the triangles opposite to the bases are separated by distance 1 and have the same x-coordinate. Therefore the absolute value of the slope of any straight line connecting a point in the triangle of G' and a point in the triangle of G'' is greater than the slope $H/W = 1 + 1/W (> 1)$ of a diagonal of the $W \times H$ rectangle. Thus, the absolute value of the slope of any edge connecting a vertex on $C_o(G')$ and a vertex on $C_o(G'')$ is greater than 1. On the other hand, by Lemma 2(b) the absolute value of the slope of every edge on $C_o(G')$ or $C_o(G'')$ is less than or equal to 1. Furthermore,

by Lemma 2(c) both the drawing of the path from v_1 to v_2 on $C_o(G^{'})$ and the drawing of the path from v_{n-1} to v_n on $C_o(G^{''})$ are x-monotone. Therefore, the straight line drawing of any edge of G connecting a vertex on $C_o(G^{'})$ and a vertex on $C_o(G^{''})$ does not intersect the drawings of $G^{'}$ and $G^{''}$. Furthermore the drawings of all these edges do not intersect each other since G is a plane graph and the drawings of the two paths above are x-monotone. Thus step 6 does not introduce any edge-intersection.

By Lemma 1 one can execute steps 1 and 2 of procedure *Draw* in linear time. By Lemma 2 one can execute steps 3 and 4 in linear time. Clearly one can execute steps 5 and 6 in linear time. Thus *Draw* runs in linear time. □

3 Drawing $G^{'}$

In this section, we show how to find a drawing of $G^{'}$ satisfying the conditions (a), (b) and (c) in Lemma 2. It suffices to decide only the coordinates of all vertices of $G^{'}$, because one can immediately find a straight line drawing from the coordinates.

We first define some terms. Let $\Pi = (v_1, v_2, \cdots, v_n)$ be a 4-canonical ordering of G. For any two vertices $v_i, v_j \in V$, we write $v_i \prec v_j$ iff $1 \le i < j \le n$, and write $v_i \preceq v_j$ iff $1 \le i \le j \le n$. We will show later that the following lemma holds.

Lemma 3. If (u, v) is an edge in $G^{'}$ and $u \preceq v$, then the y-coordinates of vertices u and v decided by our algorithm satisfy $y(u) \le y(v)$.

We say that a vertex u in a graph G is *a smaller neighbor* of v if u is a neighbor of v and u is smaller than v, that is $u \prec v$. Similarly, we say that u is *a larger neighbor* of v if u is a neighbor of v and $u \succ v$. The smallest one among the neighbors of vertex v is called *the smallest neighbor* of v, and is denoted by $w_s(v)$. We often denote $w_s(v)$ simply by w_s. Let $3 \le k \le n$, and let $C_o(G_{k-1}) = w_1, w_2, \cdots, w_m$, where $w_1 = v_1$ and $w_m = v_2$. Since G is internally triangulated, all the smaller neighbors of v_k consecutively appear on $C_o(G_{k-1})$. Thus one may assume that they are $w_l, w_{l+1}, \cdots, w_r$ for some l and r, $1 \le l < r \le m$.

We now have the following lemma.

Lemma 4. Let $\Pi = (v_1, v_2, \cdots, v_n)$ be a 4-canonical ordering of G, and let $w_l, w_{l+1}, \cdots, w_r$ be the smaller neighbors of v_k, $3 \le k \le n$. Then the following (a) and (b) hold:

(a) there is no index t such that $l < t < r$ and $w_{t-1} \prec w_t \succ w_{t+1}$; and

(b) $w_l \succeq w_{l+1} \succeq \cdots \succeq w_s \preceq \cdots \preceq w_r$, and $y(w_l) \ge y(w_{l+1}) \ge \cdots \ge y(w_s) \le \cdots \le y(w_r)$ where $w_s = w_s(v_k)$.

Proof. (a) Assume for a contradiction that there is an index t such that $l < t < r$ and $w_{t-1} \prec w_t \succ w_{t+1}$. Let $w_t = v_i$, $1 \le i \le k-1$. Since v_k is adjacent to w_{t-1}, w_t and w_{t+1} in G_k, $w_t = v_i$ is neither on $C_o(G_k)$ nor on $C_o(G)$ and hence

$3 \le i \le n-2$. Therefore by the condition (co2) of the 4-canonical ordering, w_t has at least two larger neighbors. Let v_j be the largest one among the w_t's neighbors except v_k. Then $w_t = v_i \prec v_j \ne v_k$. Clearly vertex v_j is either in the triangular face v_k, w_t, w_{t-1} of graph G_k or in the triangular face v_k, w_{t+1}, w_t. Since $w_t \prec v_j$ and $w_{t-1} \prec w_t \succ w_{t+1}$, we have $v_j \ne w_{t-1}, w_{t+1}$. Therefore v_j must be in the proper inside of one of the two faces above. Since v_j is not on $C_o(G)$, we have $3 \le j \le n-2$. Since v_j is not in G_{k-1}, we have $v_{k-1} \prec v_j$ and hence $v_k \prec v_j$. Therefore v_k is contained in G_j, and hence v_j is not on $C_o(G_j)$, contrary to the condition (co2) of the 4-canonical ordering.

(b) Since $w_s \preceq w_r$, by (a) we have $w_s \preceq w_{s+1} \preceq \cdots \preceq w_r$. Therefore by Lemma 3 we have $y(w_s) \le y(w_{s+1}) \le \cdots \le y(w_r)$. Similarly we have $y(w_l) \ge y(w_{l+1}) \ge \cdots \ge y(w_s)$.

□

We are now ready to show how to find a drawing of G'. First, we put vertices v_1, v_2, v_3 on grid points $(0,0), (2,0)$ and $(1,1)$ so that G_3 is drawn as an isosceles right-angled triangle. Clearly the conditions (b) and (c) in Lemma 2 hold for G_3. Next, for each k, $4 \le k \le \lceil n/2 \rceil$, we decide the x-coordinate $x(v_k)$ and the y-coordinate $y(v_k)$ of v_k so that the conditions (b) and (c) in Lemma 2 hold for G_k. One may assume that the conditions hold for G_{k-1}. Let $C_o(G_{k-1}) = w_1, w_2, \cdots, w_m$, and let $w_l, w_{l+1}, \cdots, w_r$ be the smaller neighbors of v_k. Since the condition (c) of Lemma 2 holds for G_{k-1}, the drawing of the path $w_l, w_{l+1}, \cdots, w_r$ is x-monotone. Furthermore, by Lemma 4(b), we have $y(w_l) \ge y(w_{l+1}) \ge \cdots \ge y(w_s) \le \cdots \le y(w_r)$, as illustrated in Fig. 4.

We always shift a drawing of G_{k-1} to the x-direction before adding vertex v_k, as illustrated in Fig. 4. We have to determine which vertices of G_{k-1} must be shifted to the x-direction. Thus we will maintain a set $U(v_k)$ for each vertex v_k, $1 \le k \le \lceil n/2 \rceil$. This set will contain vertices located "under" v_k that need to be shifted whenever v_k is shifted. Initially, we set $U(v_k) = \{v_k\}$ for $k = 1, 2, 3$. For k, $4 \le k \le \lceil n/2 \rceil$, we set $U(v_k) = \{v_k\} \cup (\cup_{i=l+1}^{r-1} U(w_i))$. Thus all vertices in $U(v_k)$ except v_k are not on $C_o(G_k)$. The *shift operation* on a vertex w_j, denoted by $shift(w_j)$, is achieved by increasing the x-coordinate of each vertex $u \in \bigcup_{i=j}^{m} U(w_i)$ by 1 [3,4,6,9,10,11].

We then show how to decide $y(v_k)$ and $x(v_k)$. Let y_{max} be the maximum value of y-coordinates of $w_l, w_{l+1}, \cdots, w_r$, then either $y_{max} = y(w_l)$ or $y_{max} = y(w_r)$. There are the following six cases:

(i) $y(w_l) < y(w_r) = y_{max}$;
(ii) $y_{max} = y(w_l) > y(w_r)$;
(iii) $y(w_l) = y(w_r) = y_{max}$, $l < s < r$ and $y(w_{l+1}) \ne y_{max}$;
(iv) $y(w_l) = y(w_r) = y_{max}$, $l < s < r$ and $y(w_{l+1}) = y_{max}$;
(v) $y(w_l) = y(w_r) = y_{max}$ and $s = l$; and
(vi) $y(w_l) = y(w_r) = y_{max}$ and $s = r$.

We first consider the three cases (i), (iii) and (v). In these cases $y_{max} = y(w_r)$. We decide $y(v_k)$ and $x(v_k)$ as follows. We first execute $shift(w_{s+1})$, that is, we

increase the x-coordinates of all vertices $w_{s+1}, w_{s+2}, \cdots, w_m$ and all vertices under them by 1, as illustrated in Figs. 4(a), (b) and (c). We then decide

$$y(v_k) = \begin{cases} y_{max} & \text{if } y(w_{r-1}) < y_{max}; \\ y_{max} + 1 & \text{if } y(w_{r-1}) = y_{max}, \end{cases}$$

and

$$x(v_k) = x(w_s) + y(v_k) - y(w_s).$$

We denote the slope of a straight line segment uv by $slope(uv)$. Then clearly we have

$$slope(w_s v_k) = \frac{y(v_k) - y(w_s)}{x(v_k) - x(w_s)} = 1.$$

Since $y(v_k) \geq y(w_l) \geq y(w_s)$ and $x(w_l) \leq x(w_s) < x(v_k)$, we have

$$0 \leq slope(w_l v_k) = \frac{y(v_k) - y(w_l)}{x(v_k) - x(w_l)} \leq slope(w_s v_k) = 1$$

as illustrated in Figs. 4(a), (b) and (c).

If $y(w_{r-1}) < y_{max}$, then $y(v_k) = y_{max} = y(w_r)$ and hence $slope(v_k w_r) = 0$ as illustrated in Fig. 4(a). On the other hand, if $y(w_{r-1}) = y_{max}$, then $y(v_k) = y(w_r) + 1$, $x(v_k) \leq x(w_r) - 1$ and hence we have

$$-1 \leq slope(v_k w_r) = \frac{y(w_r) - y(v_k)}{x(w_r) - x(v_k)} < 0$$

as illustrated in Figs. 4(b) and (c).

The absolute slope of each straight line segment on $C_o(G_k)$ except $w_l v_k$ and $v_k w_r$ is equal to its absolute slope on $C_o(G_{k-1})$, and hence is at most 1.

Thus the condition (b) in Lemma 2 holds for G_k.

One can easily observe that the condition (c) in Lemma 2 holds for G_k.

We next consider the remaining three cases (ii), (iv) and (vi). In these cases we decide $y(v_k)$ and $x(v_k)$ in a mirror image way of the cases (i), (iii) and (v) above. That is, we execute $shift(w_s)$, and decide

$$y(v_k) = \begin{cases} y_{max} & \text{if } y(w_{l+1}) < y_{max}; \\ y_{max} + 1 & \text{if } y(w_{l+1}) = y_{max}, \end{cases}$$

and

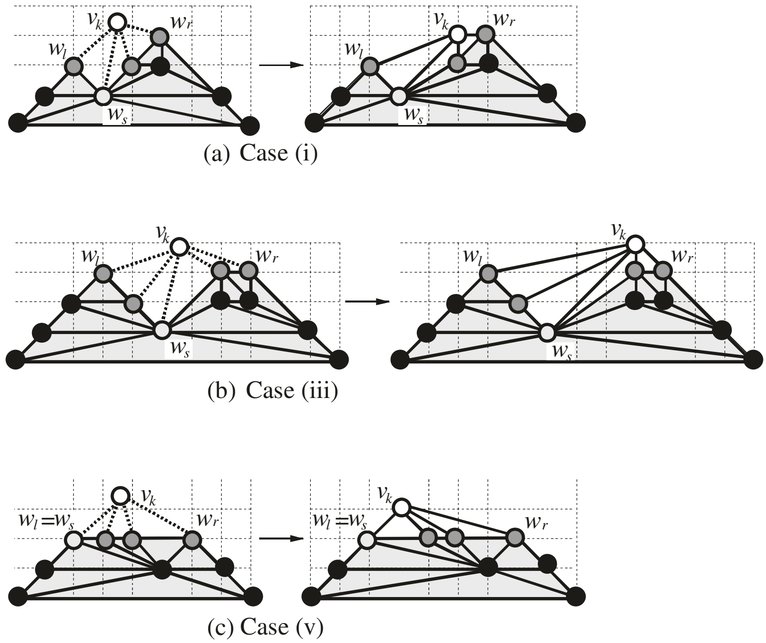
$$x(v_k) = x(w_s) - (y(v_k) - y(w_s)).$$

Then, similarly as in Case 1 above, the conditions (b) and (c) hold for G_k.

Since we decide the y-coordinate as above, Lemma 3 clearly holds.

We are now ready to prove Lemma 2.

Proof of Lemma 2 As shown above, the conditions (b) and (c) hold. Therefore the absolute value of the slope of every edge on $C_o(G')$ is at most 1, and the drawing of the path going clockwise on $C_o(G)$ from v_1 to v_2 is x-monotone.



(a) Case (i)

(b) Case (iii)

(c) Case (v)

Fig. 4. How to put v_k.

The drawing of G_3 has width 2. We execute the shift operation once when we add a vertex v_k, $4 \leq k \leq n' = \lceil n/2 \rceil$, to the drawing of G_{k-1}. Therefore the width $W^{'}$ of the drawing of $G^{'}$ is $W^{'} = 2 + (n' - 3) = \lceil n/2 \rceil - 1$. Since the conditions (b) and (c) hold, the height is at most $W^{'}/2$. Therefore $G^{'}$ is drawn in an isosceles right-angled triangle whose base has length $W^{'} = \lceil n/2 \rceil - 1$ and whose height is $H^{'} = W^{'}/2$. Obviously (v_1, v_2) is drawn as the base of the triangle. Thus the condition (a) holds.

We then show that the drawing of $G^{'}$ obtained by our algorithm is a grid drawing. Our algorithm puts each v_k, $4 \leq k \leq \lceil n/2 \rceil$, on a grid point. Clearly each edge (v_k, w_j), $l \leq j \leq r$, does not intersect any edge of G_{k-1}. Furthermore, similarly to the proof of Lemma 2 in [6], one can easily prove by induction on k that any number of executions of the shift operation for G_{k-1} introduce no edge-intersection in G_{k-1}. Thus our algorithm obtains a grid drawing of $G^{'}$.

All operations in our algorithm except the shift operation can be executed total in time $O(n)$. A simple implement of the shift operation takes time $O(n)$, and our algorithm executes the shift operation at most $\lceil n/2 \rceil$ times. Therefore a straightforward implementation would take time $O(n^2)$. However, using a data structure in [6] representing the sets $U(w_i)$, $1 \leq i \leq m$, one can implement the shift operation so that the total time required by the operation is $O(n)$.

Thus our algorithm finds a drawing of $G^{'}$ in time $O(n)$. $\square$

References

1. G. Di Battista, P. Eades, R. Tamassia and I. G. Tollis, *Automatic graph drawing: an annotated bibliography*, Computational Geometry: Theory and Applications, 4, 235-282 (1994).
2. G. Di Battista, P. Eades, R. Tamassia and I. G. Tollis, *Graph Drawing*, Prentice Hall, NJ (1998).
3. M. Chrobak and G. Kant, *Convex grid drawings of 3-connected planar graphs*, International Journal of Computational Geometry and Applications, 7, 211-223 (1997).
4. M. Chrobak and S. Nakano, *Minimum-width grid drawings of plane graphs*, Computational Geometry: Theory and Applications, 10, 29-54 (1998).
5. N. Chiba, K. Onoguchi and T. Nishizeki, *Drawing planar graphs nicely*, Acta Informatica, 22, 187-201 (1985).
6. M. Chrobak and T. Payne, *A linear-time algorithm for drawing planar graphs on a grid*, Information Processing Letters, 54, 241-246 (1995).
7. N. Chiba, T. Yamanouchi and T. Nishizeki, *Linear algorithms for convex drawings of planar graphs*, in Progress in Graph Theory, J. A. Bondy and U. S. R. Murty (eds.), Academic Press, 153-173 (1984).
8. I. Fáry, *On straight lines representation of plane graphs*, Acta. Sci. Math. Szeged, 11, 229-233 (1948).
9. H. de Fraysseix, J. Pach and R. Pollack, *How to draw a planar graph on a grid*, Combinatorica, 10, 41-51 (1990).
10. X. He, *Grid embedding of 4-connected plane graphs*, Discrete & Computational Geometry, 17, 339-358 (1997).
11. G. Kant, *Drawing planar graphs using the canonical ordering*, Algorithmica, 16, 4-32 (1996).
12. G. Kant and X. He, *Regular edge labeling of 4-connected plane graphs and its applications in graph drawing problems*, Theoretical Computer Science, 172, 175-193 (1997).
13. W. Schnyder, *Embedding planar graphs on the grid*, Proc. 1st Annual ACM-SIAM Symp. on Discrete Algorithms, San Francisco, 138-148 (1990).
14. S. K. Stein, *Convex maps*, Proc. Amer. Math. Soc., 2, 464-466 (1951).
15. W.T. Tutte, *How to draw a graph*, Proc. London Math. Soc., 13, 743-768 (1963).
16. K. Wagner, *Bemerkungen zum vierfarbenproblem*, Jahresbericht der Deutschen Mathematiker-Vereinigung, 46, 26-32 (1936).

Graph Embedding with Topological Cycle-Constraints*

Christoph Dornheim

Institut für Informatik
Albert-Ludwigs-Universität
D-79110 Freiburg, Germany
dornheim@informatik.uni-freiburg.de

Abstract. This paper concerns graph embedding under topological constraints. We address the problem of finding a planar embedding of a graph satisfying a set of constraints between its vertices and cycles that require embedding a given vertex inside its corresponding cycle. This problem turns out to be NP-complete. However, towards an analysis of its tractable subproblems, we develop an efficient algorithm for the special case where graphs are 2-connected and any two distinct cycles in the constraints have at most one vertex in common.

1 Introduction

In many graph drawing applications it is important to find drawings with certain geometrical or topological properties. For instance, to support the semantics of visual languages, given subgraphs should be drawn with a predefined shape. Such user-defined requirements on the graph layout are called graph drawing constraints. Due to their wide applicability, graph drawing approaches supporting them are of recent research interest [7]. So far most of these approaches, e.g. [5], focus on drawing arbitrary graphs with some sort of geometric constraints such as arithmetic restrictions on the coordinates of vertices. However, topological constraints that are less specific than geometric constraints have gained only little attention although they prove useful in expressing qualitative information about where to place the vertices.

In this paper we consider a special sort of topological constraints which we call *cycle-constraints*, each specifying a vertex and a cycle of a given graph with the meaning that the vertex must be embedded in the interior of the corresponding cycle. As an additional restriction, the drawings we are interested in are not allowed to have any crossings. Thus, we are concerned with the problem of finding a planar embedding of a graph satisfying a given set of cycle-constraints.

After recalling in Sect. 2 basic terminology for planar graphs, we provide in Sect. 3 a combinatorial characterization of graph embeddability with cycle-constraints which easily shows the problem to be in NP. Unfortunately, like many

* This work was supported by a DFG grant of the graduate school on human and machine intelligence.

J. Kratochvíl (Ed.): GD'99, LNCS 1731, pp. 155–164, 1999.

graph drawing optimization problems, the problem also turns out to be NP-hard. On the other hand, this opens the search for relevant tractable subproblems. As a first approach we develop in Sect. 4 an $O(n^2)$ algorithm for the case where graphs are 2-connected and any two distinct cycles in the constraints have at most one vertex in common.

2 Preliminaries

Assuming familiarity with basic graph terminology, we recall in this section definitions and properties of planar graphs and their embeddings as presented e.g. in [3]. Unless otherwise stated, all graphs are finite, simple and undirected. The set of *vertices* and *edges* of a graph G are denoted $V(G)$ and $E(G)$, respectively. A cycle C in G is *induced* if any edge of G joining two vertices of C is already an edge in C. Moreover, C is *non-separating* if the number of components of G does not increase when deleting C from G.

A *plane graph* (V, E) consists of a finite set $V \subseteq \mathbb{R}^2$ and a finite set E of Jordan curves connecting two points of V such that any two points are connected by at most one curve and the interior of a curve contains no point of V and no point of any other curve. These points and curves are again called *vertices* and *edges*, respectively, since all graph notions can also be applied to plane graphs. If there is an isomorphism between an abstract graph G and a plane graph $\widetilde{G}$, we call $\widetilde{G}$ an *embedding* of G in the plane. A graph is *planar* if it has an embedding in the plane. In the sequel $\widetilde{G}$ always denotes an embedding of the graph G.

Every plane graph G divides $\mathbb{R}^2 \setminus G$ into disjoint open arcwise connected regions called *faces* of G; the unbounded one is the *outer face* and the bounded ones are the *inner faces*. If a cycle C of G is the boundary of some face, we call C a *facial cycle*; depending on whether C is the boundary of an inner or outer face, we call C an *inner* or *outer facial cycle*, respectively. The interior of C is denoted by *int* C and its closure by *Int* C; *ext* C and *Ext* C are defined analogously.

For characterizing graph embeddability with cycle-constraints we need the following notions and results concerning the cycle structure of graphs (see [8], [3]). Given a graph $G = (V, E)$ the *edge space* $\mathcal{E}(G)$ of G is the vector space over the 2-element field $\mathbb{F}_2 = \{0, 1\}$ formed by the set of all subsets of E and the sum $A \oplus B = (A \setminus B) \cup (B \setminus A)$ for $A, B \subseteq E$. The *cycle space* $\mathcal{C}(G)$ is the subspace of $\mathcal{E}(G)$ generated by the cycles of G – more precisely, by their edge sets. The elements of $\mathcal{C}(G)$ are unions of edge-disjoint cycles of G. Suppose $\widetilde{G}$ is an embedding of a 2-connected planar graph G and $\mathcal{F}'$ is the set of all facial cycles of $\widetilde{G}$. Then $\mathcal{F} = \mathcal{F}' \setminus \{F\}$ forms a basis of $\mathcal{C}(G)$ for any $F \in \mathcal{F}'$ and F is the sum of all cycles in $\mathcal{F}$. There is always an embedding of G where $\mathcal{F}$ is the set of inner facial cycles and F the outer facial cycle. Moreover, any two embeddings of G are topologically equivalent if they coincide in their outer facial cycle and their set of inner facial cycles. We can thus represent a particular 2-connected embedding up to topological equivalence just by its outer facial cycle and the set of its inner facial cycles. If in addition G is 3-connected, the facial cycles of

$\widetilde{G}$ can be identified combinatorially as the induced non-separating cycles in G. This allows representing any embedding of G just by its outer facial cycle.

Finally, we introduce a notion (as presented in [8]) which plays a fundamental role in graph embedding algorithms following the principle of path addition. Suppose H is a subgraph of G. Then an *H-component* of G is either an edge (together with its ends) in $E(G) \setminus E(H)$ joining two vertices of H or it is a connected component of $G - H$ together with all edges (and their ends) of G joining this component to H. The vertices of an H-component in H are its *vertices of attachment*. If G is 2-connected and C a cycle in G, the vertices of attachment of any C-component of G divide C into edge-disjoint paths, called *segments*. Two C-components *overlap* if not all vertices of attachment of one C-component lie in a single segment of the other C-component. The *overlap graph* of C in G is defined as the graph whose vertices are the C-components of G such that two vertices are adjacent iff the corresponding C-components overlap.

3 Cycle-Constraints

In this section we precisely define cycle-constraints and prove two general results on the problem of finding embeddings satisfying cycle-constraints: a combinatorial characterization and NP-completeness.

Definition 1. *Let G be a graph. A* cycle-constraint *is an expression of the form $x \odot C$, with $x \in G$, cycle $C \subseteq G$ and $x \notin C$. $x \odot C$ is satisfied in a plane graph $\widetilde{G}$ if $\widetilde{x} \in int\ \widetilde{C}$ holds in $\widetilde{G}$. Then, G is* $\mathcal{S}$-embeddable *for a set $\mathcal{S}$ of cycle-constraints if there is an embedding $\widetilde{G}$ of G, called* $\mathcal{S}$-embedding, *which satisfies all the cycle-constraints in $\mathcal{S}$.*

Note that cycle-constraints are topological in the sense that topological transformations of the plane preserve their validity. For the purpose of finding appropriate algorithms it could be useful to have a necessary and sufficient condition for $\mathcal{S}$-embeddability at hand where topological notions are not involved. Besides from what is given in Sect. 2, the following theorem makes use of an easily verifiable observation: if $\widetilde{G}$ is an embedding of a 2-connected planar graph G and C is a cycle of G, then the points of $\widetilde{G}$ contained in Int $\widetilde{C}$ are exactly the vertices of those inner facial cycles of $\widetilde{G}$ whose sum is C.

Theorem 1. *Let G be a graph and $\mathcal{S}$ be a set of cycle-constraints. Then, G is $\mathcal{S}$-embeddable iff there is a 3-connected planar graph G' with $G \subseteq G'$ and $V(G) = V(G')$ and a basis $\mathcal{F}$ of $\mathcal{C}(G')$ consisting of induced non-separating cycles of G' such that for any $x \odot C$ in $\mathcal{S}$: $x \in F$ for some $F \in \mathcal{F}_C$ where $\mathcal{F}_C \subseteq \mathcal{F}$ is defined by $C = \bigoplus_{F \in \mathcal{F}_C} F$.*

Proof. ($\Rightarrow$) Suppose $\widetilde{G}$ is an $\mathcal{S}$-embedding of G. Triangulating $\widetilde{G}$ in the plane leads to an embedding $\widetilde{G'}$ of a maximal planar and thus 3-connected graph G'. The inner facial cycles of $\widetilde{G'}$ are induced non-separating cycles and form a basis

$\mathcal{F}$ of $\mathcal{C}(G')$. Then, if $x \odot C$ holds in $\widetilde{G}$, and thus in $\widetilde{G'}$, x belongs to a cycle of some subset of $\mathcal{F}$ whose sum is C.

($\Leftarrow$) Let G' be the extension of G satisfying the required properties. Then there is an embedding $\widetilde{G'}$ of G' with $\bigoplus_{F\in\mathcal{F}} F$ as the outer facial cycle and all $F \in \mathcal{F}$ as inner facial cycles. According to the observation, $\widetilde{x} \in \operatorname{int} \widetilde{C}$ holds in $\widetilde{G'}$ for all $x \odot C$ in $\mathcal{S}$. Thus, we get an $\mathcal{S}$-embedding $\widetilde{G}$ by restricting $\widetilde{G'}$ to G. □

Applying Theorem 1 requires to determine the coefficients of a given cycle C with respect to a basis $\mathcal{F}$ of $\mathcal{C}(G')$ whose elements are inner facial cycles of some embedding $\widetilde{G'}$ of G'. The following property (see [6]) leads directly to an appropriate procedure working also when G' is 2-connected. If $\mathcal{F}_C$ is the subset of $\mathcal{F}$ whose elements sum up to C, the cycles of $\mathcal{F} \setminus \mathcal{F}_C$ can be ordered as $F_1, \ldots, F_k$ such that C_i is always a cycle and $C_k = C$, where $C_0 := \bigoplus_{F\in\mathcal{F}} F$ and $C_{i+1} := C_i \oplus F_i$. Note that for $F \in \mathcal{F}$ $C_i \oplus F$ is a cycle if and only if $C_i \cap F$ is a non-trivial path in G'. We therefore have to find in each step i a new $F_i \in \mathcal{F}$ so that $C_i \cap F_i$ is a non-trivial path having no edge with C in common. Obviously, this procedure needs polynomial time.

If G is already 3-connected, we can determine whether G is $\mathcal{S}$-embeddable by first finding the set $\mathcal{F}'$ of induced non-separating cycles, e.g. simply by embedding G, and then searching for some $F \in \mathcal{F}'$ satisfying with $\mathcal{F} = \mathcal{F}' \setminus \{F\}$ the condition in Theorem 1. Thus, finding $\mathcal{S}$-embeddings of 3-connected graphs needs polynomial time.

The next theorem proves the general problem of finding $\mathcal{S}$-embeddings to be NP-complete. This holds even for rather simple 2-connected graphs as defined in the proof.

Theorem 2. *The problem of deciding whether a graph G is $\mathcal{S}$-embeddable for some given set $\mathcal{S}$ of cycle-constraints is NP-complete.*

Proof. Membership in NP follows immediately from what is said above: we only have to guess a 3-connected planar supergraph G' of G and verify in polynomial time G' to be $\mathcal{S}$-embeddable.

Next we show the problem to be NP-hard by transformation from the NP-complete problem BETWEENNESS defined as (see [4]): given a finite set A and a set T of ordered triples of distinct elements from A, is there a one-to-one function $f : A \longrightarrow \{1, \ldots, |A|\}$ such that for each $(a, b, c) \in T$ either $f(a) < f(b) < f(c)$ or $f(c) < f(b) < f(a)$ holds? Let $A = \{a_1, \ldots, a_n\}$ and $T = \{t_1, \ldots, t_m\}$ with $t_i = (a_{i_1}, a_{i_2}, a_{i_3})$ be an arbitrary instance of BETWEENNESS. We associate with A the graph G_A defined by $V(G_A) = \{x, y, a_1, \ldots, a_n\}$ and $E(G_A) = \{xa_i, a_iy \mid i \in \{1, \ldots, n\}\}$. For any distinct $i, j \in \{1, \ldots, n\}$ let $C_{i,j}$ denote the cycle x, a_i, y, a_j, x. The set S_T of cycle-constraints encoding T is defined as $S_T := \{a_{i_2} \odot C_{i_1,i_3} \mid t_i = (a_{i_1}, a_{i_2}, a_{i_3}) \in T\}$. Then, obviously, G_A is S_T-embeddable iff there is some function f satisfying the requirements given above. □

4 Restricted Cycle-Constraints

Once a decision problem is proved to be NP-complete, one is left with several possibilities to cope with this problem, one of which is searching for tractable subproblems. In case of graph embeddability with cycle-constraints one can study its complexity with respect to various restrictions on graphs and cycle-constraints. For instance, concerning the connectivity number of graphs we have already determined in Sect. 3 the exact boundary between P and NP: 3-connected graphs make the problem tractable, but a smaller connectivity number leads to NP-completeness. In this section, we restrict graphs to be 2-connected and cycle-constraints to have the property that any two distinct cycles share at most one vertex. In the remainder let G be a 2-connected graph, $\mathcal{S}^0$ be a non-empty set of cycle-constraints of this sort with respect to G and S be the set of cycles occurring in $\mathcal{S}^0$.

We next present in Theorem 3 a necessary and sufficient condition for the existence of $\mathcal{S}^0$-embeddings for 2-connected graphs as well as an $O(n^2)$ algorithm (where $n = |G|$) for finding them. For technical convenience the sufficient condition of this theorem is proved at the end of this section in Proposition 2 where the algorithm is shown to be valid. As it turns out, $\mathcal{S}^0$-embeddability can be characterized with some similarity to the well-known theorem (see [8]) that a graph is planar iff the overlap graph of each cycle is bipartite. For that we first introduce the notion of an $\mathcal{S}^0$-partition.

Definition 2. *Let $C \in S$ and G^C be the overlap graph of C. An $\mathcal{S}^0$-*partition *of G^C is a pair (V_I, V_O) of subsets V_I, V_O of $V(G^C)$ with $V_I \,\dot\cup\, V_O = V(G^C)$ and*

1. *if $XY \in E(G^C)$, then $X \in V_I, Y \in V_O$ or $X \in V_O, Y \in V_I$*
2. *if $x \odot C$ in $\mathcal{S}^0$ and $x \in Y \in G^C$, then $Y \in V_I$.*

Theorem 3. *G is $\mathcal{S}^0$-embeddable iff G is planar and there are some $C_r \in S$ and $\mathcal{S}^0$-partitions (V_I, V_O) of the overlap graph G^C for any cycle $C \in S$ such that for each $C \neq C_r$: $Y \in G^C$ with $C_r \subseteq Y$ is in V_O.*

Proof. ($\Rightarrow$) Let $\widetilde{G}$ be an $\mathcal{S}^0$-embedding of G and select some $C_r \in S$ which is contained in Ext $\widetilde{C}$ of $\widetilde{G}$ for any other cycle $C \in S$. Define for each $C \in S$ an $\mathcal{S}^0$-partition (V_I, V_O) of G^C by $Y \in V_I$ if $Y \subseteq$ Int $\widetilde{C}$ in $\widetilde{G}$, $Y \in V_O$ otherwise.

($\Leftarrow$) (see Proposition 2) □

4.1 Decomposition Trees

The fundamental idea of the embedding algorithm to be developed is to decompose G completely into certain subgraphs that support the control of the $\mathcal{S}^0$-constraints during the embedding process. These subgraphs build the decomposition tree of G. To introduce decomposition trees precisely, we first define the concept of a reduced C-component of a cycle C. The following definition is illustrated in Fig. 1.

Definition 3. *Let B_C be a C-component of G for $C \in S$. If $C' \subsetneq B_C$ for some $C' \in S$, let $B_{C,C'}$ be the C'-component of $B_C \cup C$ which contains C. Then,*

$$B^r_C := \begin{cases} B_C \cap \bigcap_{C' \in S, C' \subsetneq B_C} (B_{C,C'} \cup C') & \text{if } B_C \text{ contains a cycle of } S \\ B_C & \text{otherwise} \end{cases}$$

is called the reduced C-component *of G in B_C with respect to S.*

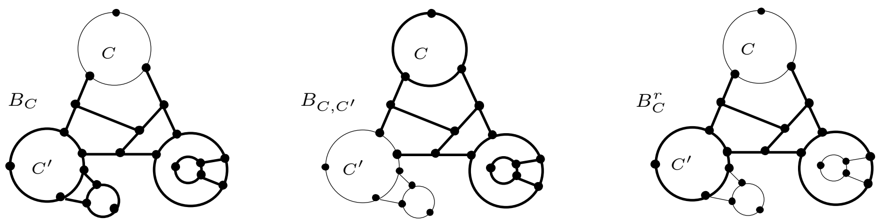


Fig. 1. Construction of the reduced C-component B^r_C from C-component B_C

After removing all C'-components not containing C for cycles $C' \subsetneq B_C$, the reduced C-component B^r_C in B_C has the property of having exactly one C'-component in $B^r_C \cup C$ for each $C' \subsetneq B^r_C$. Alternatively, we could have defined B^r_C using this property:

Proposition 1. *Let B_C be a C-component of G for $C \in S$ and $\mathcal{B}$ be the set of subgraphs $B \subsetneq B_C$ having only one C'-component in $B \cup C$ for all $C' \in S$ with $C' \subsetneq B$. Then, B^r_C is exactly the maximal graph in $\mathcal{B}$.*

Note that, by Proposition 1, any C'-component in $B^r_C \cup C$ is already reduced and, more interestingly, any embedding of $B^r_C \cup C$, if any, is completely contained either in Int $\widetilde{C'}$ or Ext $\widetilde{C'}$ for each $C' \subsetneq B^r_C \cup C$. This feature of planar reduced components is particularly useful in the embedding procedure.

Corollary 1. *If $B^r_C \cup C$ is planar, then any $C' \in S$ with $C' \subsetneq B^r_C \cup C$ is a facial cycle in every embedding of $B^r_C \cup C$.*

If $C' \in S$ is a cycle in B^r_C, each C'-component not including C is removed in B^r_C. However, it contains itself a reduced component as well as the components removed in that one. Thus, the graph G can be recursively decomposed into reduced components with respect to cycles in S. The decomposition tree is a directed tree with root C containing the complete structure of this decomposition process. Fig. 2 (b) and (c) (ignore the labels) show two different decomposition trees of the graph given in (a).

Definition 4. *Let C be a cycle in S. For any $C' \in S$ with $C' \subsetneq B^r_C$ let $G_{B^r_C,C'}$ denote the union of C' and all C'-components in $B_C \cup C$ not containing C. If*

S^r_C is the set of all reduced C-components in G, the decomposition tree *of G with respect to C and S is defined as*

$$T_{G,C} := \begin{cases} \{(C,B) \mid B \in S^r_C\} \cup \{(B,C') \mid C' \subseteq B \in S^r_C\} \cup \\ \bigcup_{C' \subseteq B \in S^r_C} T_{G_{B,C'},C'} & \text{if } G \neq C \\ \{G\} & \text{if } G = C. \end{cases}$$

Since the subgraphs $G_{B,C'}$ are again 2-connected, $T_{G,C}$ is well-defined. By definition, all cycles of S are vertices in $T_{G,C}$. Moreover, G is completely decomposed, i.e. $G = \bigcup_{t \in T_{G,C}} t$. This follows inductively from $B_C = B^r_C \cup G_{B^r_C,C_1} \cup \cdots \cup G_{B^r_C,C_k}$ if $C_1, \ldots, C_k$ are all cycles of S in B^r_C.

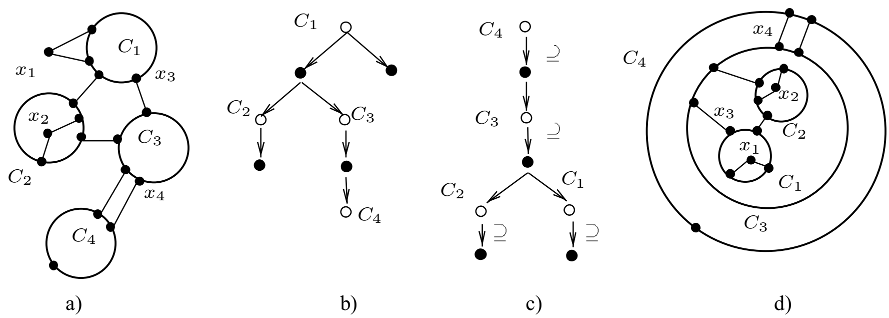


Fig. 2. (a) A graph G with $\mathcal{S}^0 = \{x_i \odot C_i \mid i \in \{1,2,3,4\}\}$; (b) decomposition tree T_{G,C_1}; (c) labelled decomposition tree T^l_{G,C_4}; (d) $\mathcal{S}^0$-embedding of G

The recursive definition of $T_{G,C}$ in Definition 4 leads directly to an algorithm for generating the decomposition tree of G with respect to a cycle C of S. Instead of using Definition 3 to compute the reduced components, one can apply the following informally given procedure. Suppose B_C is a C-component of cycle C and A the set of its vertices of attachment. Start a depth-first search of B_C beginning at some vertex of A. Whenever a cycle $C' \in S$, $C' \neq C$, appears for the first time, determine all C'-components in B_C not including all vertices of A and remove them from B_C. The union of these C'-components together with C' is actually $G_{B^r_C,C'}$. When the search is finished, the subgraph denoted by B_C is now the reduced C-component. Using Proposition 1 one can easily verify that the whole procedure computes $T_{G,C}$ correctly. Moreover, if G is planar, it only needs $O(n^2)$ time where $n = |G|$ since the number of edge-disjoint cycles in G cannot exceed n.

4.2 Embedding Algorithm

Provided with Theorem 3 and the concept of decomposition trees, we are now able to describe an $O(n^2)$ algorithm for constructing an $\mathcal{S}^0$-embedding of G if

G has any. Instead of a particular drawing the algorithm generates a formal representation of an $\mathcal{S}^0$-embedding consisting of its outer facial cycle and the set of its inner facial cycles.

Step 1. First, we must check G to satisfy the necessary condition of Theorem 3 for being $\mathcal{S}^0$-embeddable and thereby determine the particular cycle $C_r \in S$. That means we test planarity and determine the overlap graph G^C of any cycle $C \in S$. For each $C \in S$ we consider the components H of G^C: find an $\mathcal{S}^0$-partition (V_I, V_O) of H, if any, and define $M_H = V_O$ if $x \in Y \in H$ for some constraint $x \odot C$, otherwise $M_H = V_I \cup V_O$. Let M_C be the union of all sets M_H for components H of G^C; M_C contains the C-components that can be embedded outside from C and therefore are candidates for the choice of C_r. Finally, we have to find a $C_r \in S$ which is in M_C of any other cycle $C \in S$. The cycle C_r satisfies the condition of Theorem 3 and is used furthermore to construct an $\mathcal{S}^0$-embedding of G. If there is no such cycle or some H has no $\mathcal{S}^0$-partition, we can stop trying to embed G. Since the number of cycles in S cannot exceed n, this procedure needs $O(n^2)$ time.

Step 2. Having found the cycle C_r, we compute the decomposition tree T_{G,C_r} with root C_r as described in Sect. 4.1. Next we encode the $\mathcal{S}^0$-constraints directly into T_{G,C_r} by labelling its edges. Suppose $x \odot C$ is a constraint in $\mathcal{S}^0$. Then, by definition of C_r, there must be a path $C, B, \ldots, D$ from C to some D in T_{G,C_r} with $x \in D$. We thus label the edge (C, B) with $\supseteq$ since in any $\mathcal{S}^0$-embedding of G the reduced C-component B must be in the interior of C. When all constraints in $\mathcal{S}^0$ are encoded in this way, the labelled decomposition tree is denoted by T^l_{G,C_r}; an example is shown in Fig. 2 (c).

Step 3. In the last step we obtain with algorithm *embedding* given in Fig. 3 a representation of an $\mathcal{S}^0$-embedding. Roughly speaking, this algorithm is of the same spirit as the path addition procedure originally presented by Demoucron et al. [2] and also briefly described in [1]. As G and C_r satisfy the necessary condition of Theorem 3, algorithm *embedding* generates a set $\mathcal{F}$ of cycles of G and a set $\mathcal{F}^I$ of sets $\mathcal{F}^I_C \subseteq \mathcal{F}$ for each $C \in S$ such that for any $F \in \mathcal{F}$ contained in no $\mathcal{F}^I_C$ there is some $\mathcal{S}^0$-embedding of G with F as the outer facial cycle and $\mathcal{F} \setminus \{F\}$ as the set of inner facial cycles.

We now explain the algorithm in more detail. The parameters C, G', $\mathcal{F}$ and $\mathcal{F}^I$ have the following meaning: C is a cycle in S and the currently visited vertex of the depth-first search through T^l_{G,C_r}; G' is the subgraph of G processed so far ($C \subseteq G'$); $\mathcal{F}$ is a set of facial cycles of some embedding of G' and $\mathcal{F}^I$ a set of sets $\mathcal{F}^I_{C'} \subseteq \mathcal{F}$ for each visited cycle $C' \in T^l_{G,C_r}$. The embedding procedure must be started with the call *embedding*$(T^l_{G,C_r}, C_r, C_r, \{C_r\}, \{\mathcal{F}^I_{C_r}\})$ where $\mathcal{F}^I_{C_r} = \{C_r\}$. Suppose C, G', $\mathcal{F}$ and $\mathcal{F}^I$ are the current variables. Algorithm *embedding* expand $\mathcal{F}$ and $\mathcal{F}^I$ with all reduced C-components B. In choosing which B to take first, those B are preferred that can be attached to only one facial cycle

Algorithm *embedding* $(T^l_{G,C_r}, C, G', \mathcal{F}, \mathcal{F}^I)$

Determine the set $\mathcal{M}$ of B_i where (C, B_i) is an edge in T^l_{G,C_r};
while $\mathcal{M} \neq \emptyset$ **do**
  for all $B_i \in \mathcal{M}$ **do**
    Determine the set $\mathcal{F}_{B_i}$ of cycles from $\mathcal{F}$
    containing all vertices of $C \cap B_i$;
    if (C, B_i) has label $\supseteq$ **then** $\mathcal{F}_{B_i} \leftarrow \mathcal{F}_{B_i} \cap \mathcal{F}^I_C$;
  if $|\mathcal{F}_{B_i}| = 1$ for some $B_i \in \mathcal{M}$
   then $B \leftarrow B_i$ and $F \in \mathcal{F}_{B_i}$
   else $B \leftarrow B_i$ for some B_i and $F \in \mathcal{F}_{B_i}$;
  $G' \leftarrow G' \cup B$;
  Determine the set $\mathcal{F}'$ of facial cycles of an embedding of $F \cup B$;
  if $\mathcal{F} = \{C_r\}$ **then** $\mathcal{F} \leftarrow \mathcal{F}'$
        else $\mathcal{F} \leftarrow (\mathcal{F} \cup \mathcal{F}') \setminus \{F\}$;
  for all $\mathcal{F}^I_{C'} \in \mathcal{F}^I$ **do**
    if $F \in \mathcal{F}^I_{C'}$ **then** $\mathcal{F}^I_{C'} \leftarrow (\mathcal{F}^I_{C'} \cup \mathcal{F}') \setminus \{F\}$;
  for all C' with (B, C') in T^l_{G,C_r} **do**
    $\mathcal{F}^I_{C'} \leftarrow \{C'\}$;
    $\mathcal{F}^I \leftarrow \mathcal{F}^I \cup \{\mathcal{F}^I_{C'}\}$;
  $\mathcal{M} \leftarrow \mathcal{M} \setminus \{B\}$;
for all C' with $\{C'\} \in \mathcal{F}^I$ **do**
  embedding$(T^l_{G,C_r}, C', G', \mathcal{F}, \mathcal{F}^I)$;
return G', $\mathcal{F}$, $\mathcal{F}^I$;

Fig. 3. Algorithm *embedding* for generating facial cycles of an $\mathcal{S}^0$-embedding

of $\mathcal{F}$. Otherwise an arbitrary reduced C-component B and one of its two facial cycles F it can be attached to are chosen. If the edge from C to B in T^l_{G,C_r} is labelled, B must be attached to a cycle in $\mathcal{F}^I_C$. Then the facial cycles of an embedding of $B \cup F$ are determined (note that G was checked to be planar). By Corollary 1, all cycles of S contained in $B \cup F$ are facial cycles. Thus we must update $\mathcal{F}$ and all $\mathcal{F}^I_{C'}$ in $\mathcal{F}^I$ by replacing F with all new facial cycles of $B \cup F$ except F itself. Moreover, $\mathcal{F}^I$ must be extended by the new sets $\mathcal{F}^I_{C'}$ for all $C' \in S$ with $C' \subseteq B$. After attaching all B, the algorithm is applied in a depth-first manner to all cycles C' which are successors of B in T^l_{G,C_r}.

To establish the validity of this algorithm in the next proposition, let for the ith procedure call, $i = 1, \ldots, k := |S|$, C_i denote the variable C and $G'_i, \mathcal{F}_i, \mathcal{F}^I_i$ denote the variables $G', \mathcal{F}, \mathcal{F}^I$, respectively, at the time when the while-loop ends.

Proposition 2. *Suppose G is planar and the overlap graph G^C of any cycle $C \in S$ has an $\mathcal{S}^0$-partition (V_I, V_O) such that for $C \neq C_r$: $Y \in G^C$ with $C_r \subseteq Y$ is in V_O. Then, algorithm* embedding *terminates with $G'_k = G$, $\mathcal{F}_k$ and $\mathcal{F}^I_k$ such that for any $F_0 \in \mathcal{F}_k$ contained in no $\mathcal{F}^I_C$ of $\mathcal{F}^I_k$ there is some $\mathcal{S}^0$-embedding of G with F_0 as the outer facial cycle and $\mathcal{F}_k \setminus \{F_0\}$ as the set of inner facial cycles.*

Proof. In the sequel we abbreviate the union of all cycles in $\mathcal{F}_C^I \in \mathcal{F}_i$ by H_C^i. Then the proposition can be proved by showing for any $i \in \{1, \ldots, k\}$:

1. $\mathcal{F}_i$ is the set of facial cycles of some embedding of G'_i
2. $C = \bigoplus_{F \in \mathcal{F}_C^I} F$ for any $\mathcal{F}_C^I \in \mathcal{F}_i^I$
3. $B \subseteq H_{C_j}^i$, $1 \le j \le i$, for each successor B of C_j with labelled edge (C_j, B)
4. $C_r \not\subseteq H_C^i$ for any $C \in S$, $C \ne C_r$, with $C \subseteq G'_i$.

Let $i = 1$, i.e. $C_1 = C_r$. Since G^{C_r} has an $\mathcal{S}^0$-partition (V_I, V_O), the algorithm generates $G'_1, \mathcal{F}_1, \mathcal{F}_1^I$ by stepwise adding the successors B of C_r such that for some $\mathcal{S}^0$-partition (V'_I, V'_O) of G^{C_r} (which may be different from (V_I, V_O)): $B \subseteq Y \in V'_I$ iff $B \subseteq H_{C_r}^1$. Thus 3) is satisfied. By Corollary 1, for each B and F chosen in the algorithm any cycle $C \in S$ in $B \cup F$ belongs to the set $\mathcal{F}'$ of facial cycles of some embedding of $B \cup F$ found by the algorithm. This and the way how $\mathcal{F}_1$ and $\mathcal{F}_1^I$ are updated ensure 1), 2) and 4) to be valid.

Suppose $i > 1$. By assumption, G^{C_i} has an $\mathcal{S}^0$-partition (V_I, V_O) with $Y \in V_O$ if $C_r \subseteq Y$. Since 4) holds for $i - 1$, the algorithm extends $G'_{i-1}, \mathcal{F}_{i-1}, \mathcal{F}_{i-1}^I$ to $G'_i, \mathcal{F}_i, \mathcal{F}_i^I$ by adding the successors B of C_i such that for some $\mathcal{S}^0$-partition (V'_I, V'_O) of G^{C_i} with $Y \in V'_O$ if $C_r \subseteq Y$: $B \subseteq Y \in V'_I$ iff $B \subseteq H_{C_i}^i$. This shows 3); and 1), 2) and 4) can be proved analogously to what is said above. □

We finally note that this procedure needs only $O(n^2)$ time because the number of calls is $|S| \le n$ and each while-loop needs $O(n)$ time.

Acknowledgments

The author would like to thank Bernhard Nebel and Jochen Renz for their helpful comments.

References

1. J.A. Bondy, U.S.R. Murty: *Graph Theory with Applications*, MacMillan, 1976
2. G. Demoucron, Y. Malgrange, R. Pertuiset: Graphes planaires: reconnaissance et construction de représentations planaires topologiques, *Revue Française de Recherche Opérationelle*, 8: 33-47, 1964
3. R. Diestel: *Graph Theory*, Springer, 1997
4. M.R. Garey, D.S. Johnson: *Computers and Intractability – A Guide to the Theory of NP-Completeness*, Freeman, 1979
5. W. He, K. Marriott: Constrained graph layout, in S. North (ed.), *Graph Drawing*, Proceedings of GD'96, LNCS 1190, 217-232, 1997
6. M.M. Syslo: An efficient cycle vector space algorithm for listing all cycles of a planar graph, *SIAM Journal on Computing*, 10,4: 797-808, 1981
7. R. Tamassia: Constraints in graph drawing algorithms, *Constraints*, 3: 87-120, 1998
8. C. Thomassen: Planarity and duality of finite and infinite graphs, *Journal of Combinatorial Theory, Series B*, 29: 244-271, 1980

Embedding Vertices at Points: Few Bends Suffice for Planar Graphs

Michael Kaufmann and Roland Wiese

Wilhelm-Schickard-Institut für Informatik, Universität Tübingen,
Sand 13, 72026 Tübingen, Germany
{mk,wiese}@informatik.uni-tuebingen.de

Abstract. The existing literature gives efficient algorithms for mapping trees or less restrictively outerplanar graphs on a given set of points in a plane, so that the edges are drawn planar and as straight lines. We relax the latter requirement and allow very few bends on each edge while considering general plane graphs. Our results show two algorithms for mapping four-connected plane graphs with at most one bend per edge and for mapping general plane graphs with at most two bends per edge. Furthermore we give a point set, where for arbitrary plane graphs it is NP-complete to decide whether there is an mapping such that each edge has at most one bend.

1 Introduction

We consider a problem that already has some tradition in computational geometry and graph drawing. Originally, the problem was how to map a given tree T of n vertices at a given set of points S in the plane such that the edges can be drawn straightline and without any crossings. Variants of this problem have been explored, either with or without keeping the position of one specific node fixed [13,10,2].

Generalizing the graph class, but still using the required straightline planar drawing, Gritzmann et al. [9] proved that outerplanar graphs can be drawn without any bends. In the consequent papers [3] and [1], efficient implementations have been developed. The latest result in [1] is an $O(n \cdot \log^3 n)$ time algorithm to find a straightline drawing for such a graph. Astonishingly enough, the case for more general graphs has not been considered systematically. It is at exactly this point that we start.

Another similar scenario has been considered by Pach and Wenger [12]. Here the mapping of the vertices to the points is already fixed. The authors prove that $O(n)$ bends per edge are sufficient and that we can not expect to significantly improve the worst case bound for the maximum number of bends per edge.

We consider just the first scenario where the mapping of the vertices to the points is not yet fixed. On the other hand, we preserve the given planar embedding of the graph. In the next section, a simple scheme is presented, that provides drawings with at most one bend per edge for a large class of graphs.

J. Kratochvíl (Ed.): GD'99, LNCS 1731, pp. 165–174, 1999.

Next we generalize the technique so that it will work for any planar graph and produces drawings with at most two bends per edge. In section 4, we give a class of graphs and a set of points, where we can prove that there is at least one edge with two bends. Finally, we extend the techniques developed so far to a simple proof to the expected NP-completeness result, namely to decide whether a drawing can be found where each edge has at most one bend.

Throughout this paper, we deal with triangulated embedded (plane) graphs, only in the last section do we discuss a more general case. Of course, we always require the drawings to be planar, even if it is not explicitly stated.

2 A Basic Technique

In this section, we present the basic technique for the mapping. Let $G = (V, E)$ be any plane graph with a hamiltonian cycle C, such that C has at least one edge, say e, on the outer face of G. We call such a property 'external hamiltonicity' and a corresponding cycle 'external hamiltonian'. Let S be any set of points $p_1, p_2, ..., p_n$ with $p_i = (x_i, y_i)$. Assume the plane is rotated in such a way to make the x-coordinates of the points pairwise different. Furthermore assume that the points are ordered with increasing x-coordinates. Now we map the hamiltonian cycle C to the points $p_1, ..., p_n$, so that the edge $e = (v, w)$, is assigned in such a way that $v = p_n$ and $w = p_1$. All edges on C with the exception of e can be drawn as a straight line so that they extend monotonically in x-direction. The edge e is drawn from the rightmost point p_n to the leftmost point p_1 with one bend located at a place existing very high above all the other points. Nevertheless we choose the segments of e such that their slopes are the same and they are cone-shaped. The slopes of e is determined by the maximal slope of the edges on the hamiltonian path $C - e$. This also determines the place where the bend of e is located. The remaining edges are drawn each with exactly one bend such that all the segments have the same slopes as the segments of e. They will run in parallel. This is more because of aesthetic reasons and to simplify the arguments about avoiding some possible crossings. The edges inside of C are drawn above the polygonal chain $C - e$, and the edges outside are drawn below. Edges inside and outside of C do not cross since they are separated by $C - e$. Any two edges inside of C do not intersect because of planarity. The same holds for the edges outside. The slopes for the segments of the edges not in C have been chosen large enough such that edges in C cannot interfer with edges not in C. This roughly indicates that planarity is preserved.

Note that the area may be much larger than the area R occupied by the point set. More precisly, let us assume that the minimal enclosing rectangle R is a square of width W and δ is the minimal distance in x-direction between any two points. A short analysis shows that the resulting height might be at least $W/\delta \cdot W$, while the width remains the same. This means that if we assume integer-coordinates ($\delta = 1$), we achieve an area of W^3 for the drawing.

Note that if we would allow 2 bends per edge, we could easily draw the edges in an orthogonal way and keep the area linear in the size of the point set.

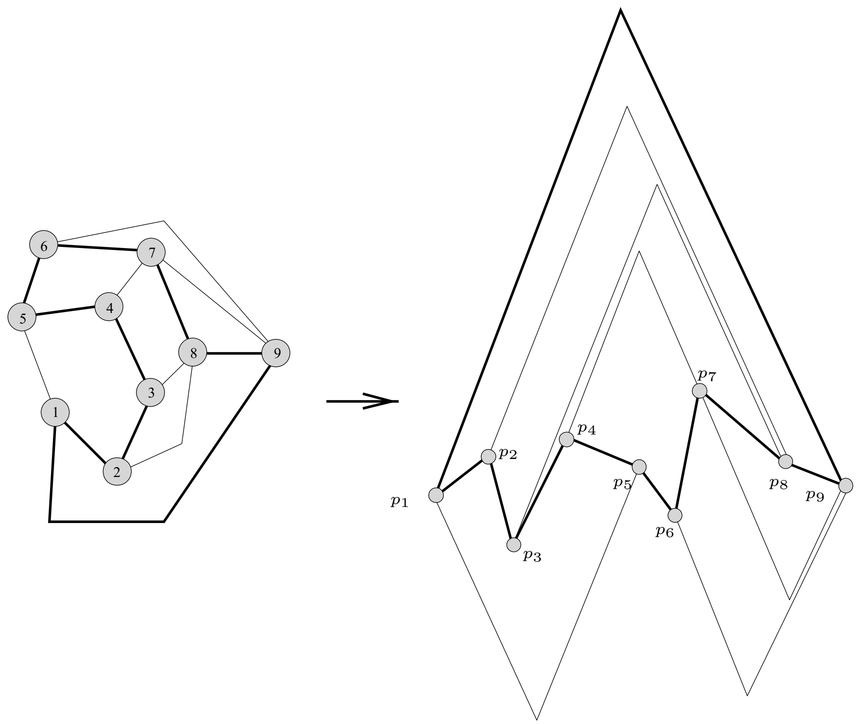


Fig. 1. The basic construction

Theorem 1. *Let S be an arbitrary set of n points and G be any plane graph with an external hamiltonian cycle and n vertices. Then G can be drawn planar with a mapping of the vertices to the points such that each edge has at most one bend.*

3 The General Case

In order to apply the technique above we need to find an external hamiltonian cycle, namely a hamiltonian cycle including an edge on the outer face. Testing all possible edges $e = (v, w)$ on the outer face, we could request a hamiltonian path, which is a well-known NP-complete problem even on planar graphs. On the other hand, we know of a linear-time algorithm to find external hamiltonian cycles by Chiba and Nishizeki [5], if the graph is four-connected. Since the graphs we consider are triangulated, the problematic cases appear if there are separating triangles, namely cycles of length 3 which do not circumscribe single faces. Only such graphs may not contain external hamiltonian cycles.

First of all, we give a reduction to the four-connected case which will finally lead to drawings with at most two bends per edge. Then, in the following section we present a small plane graph without any external hamiltonian cycle and a

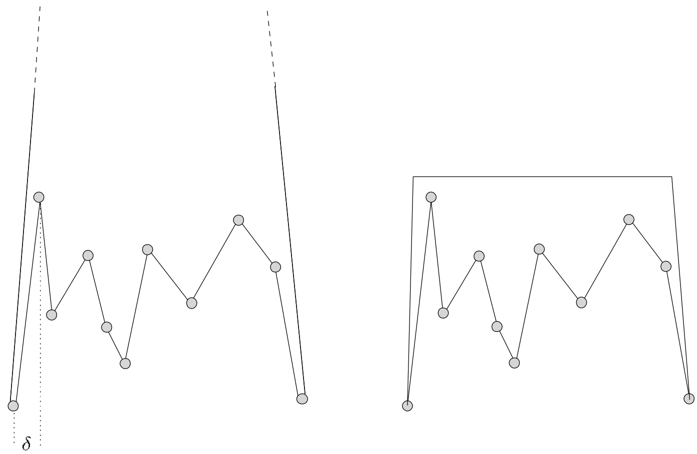


Fig. 2. Wasting and saving some area

point set, and we prove that any planar drawing of this graph on this point set must have at least one edge with 2 bends. This indicates that our simple technique is reasonably good and it will not normally be beaten by other algorithms with respect to the maximal number of bends per edge.

Assume G is a plane triangulated graph which is **not** four-connected. Let $e = (v, w)$ be an edge of any particular separation triangle which clearly exists. Edge e lies near two triangular faces (v, w, s) and (v, w, t). We destroy those triangles by inserting a dummy vertex z on e and connecting z by edges to the vertices v, w, s and t. Note that by each single operation, the number of separation triangles decreases and no such triangles are created anew. The dummy vertices z do not appear in any separating triangle. We perform this operation until all separating triangles are destroyed. The separating triangles can be efficiently found by the algorithm of Chiba and Nishizeki [4]. Then the new graph G' is four-connected.

We now apply the technique described above to G'. The only modification is the handling of the dummy vertices z. Figure 3 gives an example.

Let C' be the external hamiltonian cycle as found by the algorithm of Chiba and Nishizeki. Clearly, C' visits z, and immediately before and afterwards, it visits two vertices $a, b \in \{v, w, s, t\}$. We place a new dummy point p_z exactly between the points assigned to a and b.

Then the graph can be drawn as described above. Finally, we remove the edges (s, z) and (t, z) and join the (at most) two segments of (v, z) and (z, w). This immediately gives a drawing with at most 3 bends per edge, since there is at most one dummy vertex on each edge.

Lemma 1. *Given an arbitrary plane graph with n vertices and a set of n points in the plane. In time $O(n \log n)$, we can find a mapping of the vertices to the points, so that the edges can be drawn planar and with at most 3 bends each.*

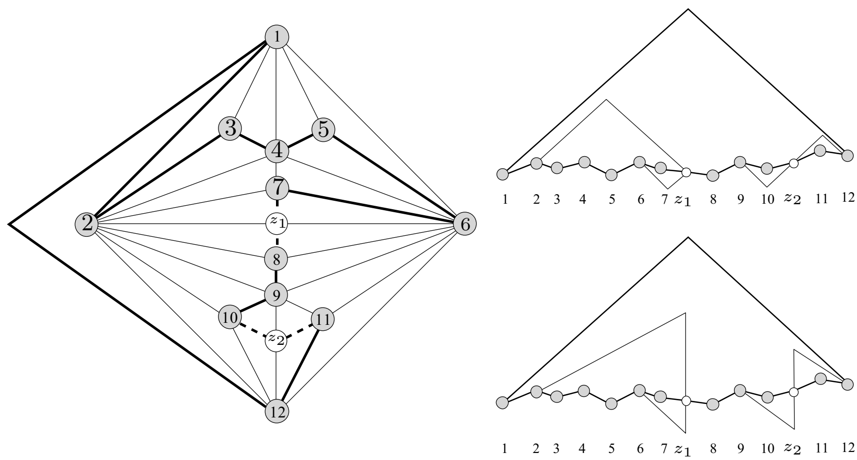


Fig. 3. An example for the construction of graphs without external hamiltonian cycle. Vertices z_1 and z_2 are dummies. They arise when destroying the separating triangles. The figures to the right indicate the solutions with three and two bends respectively.

Markus Eiglsperger suggested a way of saving one bend (out of three) by drawing some of the segments of the edges vertically. In the third part of Figure 3, we indicate the idea.

Lemma 2. *Given a solution with at most three bends for each edge constructed by the algorithms above, we can modify the drawing so that it remains planar and the maximal number of bends is two. The used area might grow exponentially.*

Proof: Let P be the designated hamiltonian path along the points $p_1, \dots p_n$ such that the edges (p_i, p_{i+1}) are drawn as a straight line. P induces a partition of the drawing plane into an upper and a lower part. Note that for each edge e with two or three bends there is a dummy vertex d_e placed on an the edge (p_i, p_{i+1}) where the edge e crosses path P. Following the construction above, it is clear that each edge crosses P once at most, hence the two segments of e incident to the dummy vertex d_e may be able to be drawn vertically. We discuss now the implications of such operations:

We consider just the section in the upper part of the drawing, the lower part is handled analogously. Let e be the edge under consideration with segments s_1 and s_2 where s_2 ends at dummy vertex d_e. Let α_1 and α_2 be the angles indicating the slopes of the segments as shown in figure 4.

Stretching s_1 such that α_1 remains the same, the angle α_2 increases to 90^o and the segment s_2 becomes vertical. We will call it s_2' now. Planarity is eventually violated if there are some segments s with angle β crossing the cone between s_2 and s_2'. We can correct this easily by rotating the segment s such that β also increases. This process is iterated if necessary. Obviously it ends after at most m steps since we only proceed from left to right and never backtrack. The proper

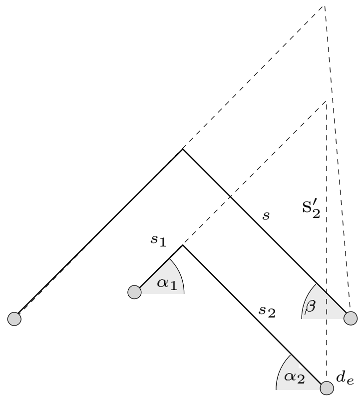


Fig. 4. The configuration before and after rotating segment s_2 in vertical position.

nesting of the edges (halfedges) in the upper part of the drawing ensures that for each edge only m rotations are necessary, implying a quadratic running time.

Combined with a corresponding process for the lower part we end when the segments incident to any dummy vertices are vertical and the corresponding bend is saved.

In the second part of the proof we sketch a situation where the area grows exponentially. The next figure shows two nested edges with corresponding dummy vertices on different sides (left and right).

We assume that the slopes are at 45^o to start with and the points and bends lie on integer coordinates. When we perform the modifications described above, so that the segments incident to the dummy vertices become vertical, the drawing grows by more than a factor of two.

Now assume that we have $n/2$ of such pairs nested, as indicated in the next figure. Consider the i-th pair from the inside. The drawing of G_{i-1} includes an axis-parallel rectangle R_i determined by the length of the vertical segments of the edges from the $i-1$-th pair. Next, we see that the two edges from pair i have to circle around this rectangle using only two bends and one vertical segment in the middle. It follows quite easily that the lengths of these segments must be quite large compared to the height of the rectangle R_{i-1} and that a new rectangle R_i of height at least twice as large as the height of R_{i-1} results. Hence, we can conclude that the height of the drawing grows exponentially, at the very least.

We conclude with a note regarding the runtime. Clearly the first part of the construction works in linear time, since we can use the linear time algorithm of Chiba/Nishizeki [4] to determine the separating triangles. Then the saving of the third bends by rotating some of the segments might cause a quadratic number of steps.

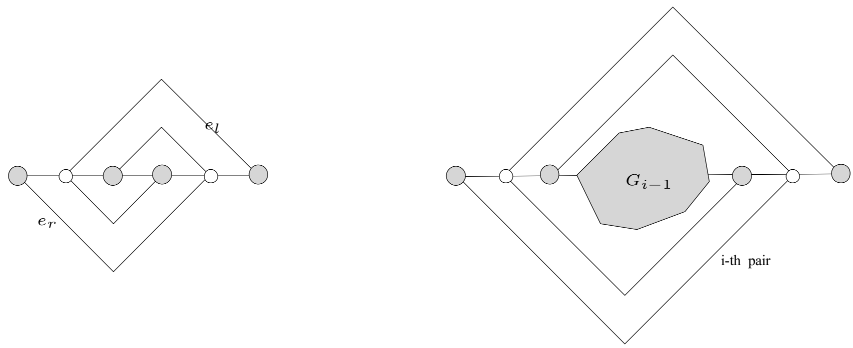


Fig. 5. The recursive definition of the graph with exponential height.

Theorem 2. *Any plane graph can be mapped on any given point set in the plane and can be drawn with at most three bends per edge in linear time and with at most two bends per edge in quadratic time.*

4 The Lower Bound

Next, we show that this bound is optimal in the worst case. Consider the following triangulated graph discussed in the example from the last figure.

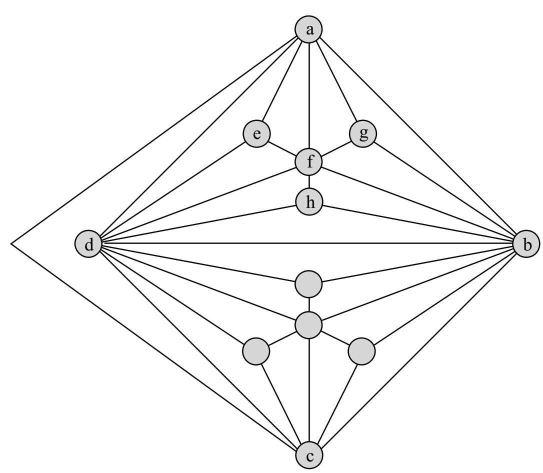


Fig. 6. The candidate for the lower bound proof.

Although there is a hamiltonian path in G, there is no external hamiltonian path. We try to map G on a set of 12 points with the same y-coordinate Y. This point set has the property that each edge with only one bend must lie completely

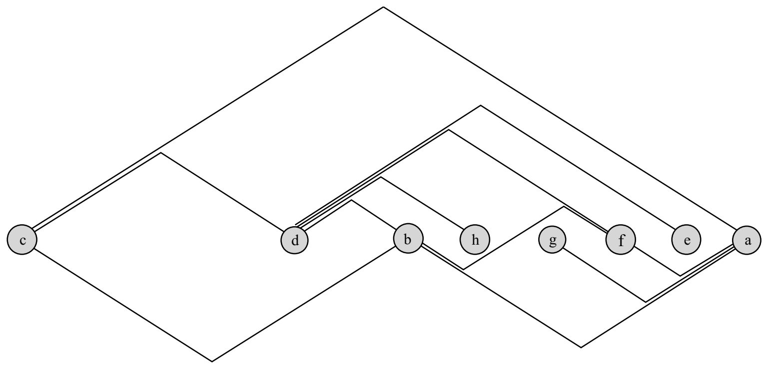


Fig. 7. Edge (b, f) needs two bends in this drawing.

above or completely below the Y-line. Any edge segment that crosses the Y-line must belong to an at-least-two bend edge.

Let $x_1, \ldots, x_n$ be the x-coordinates of the points in increasing order. Since the outer face of G is a triangle with vertices a, b, c it is clear that in any one bend drawing $\{x_1, x_{12}\} \subset \{x_a, x_b, x_c\}$ (With x_a, we mean the x-coordinate vertex a is mapped to). We examine the case where $x_1 = x_c, x_{12} = x_a$. Both of the other cases are similar or symmetric. Next we want to draw the outer face. For that we map b to $x_b, (x_1 < x_b < x_{12})$ and draw the outer face edges such that (a, c) bends above the Y-line and $(a, b), (b, c)$ bend below. Next we draw the edges of the triangles b, d, a and b, c, d. Without loss of generality, we map vertex d to $x_d, (x_a < x_d < x_b)$ and draw $(a, d), (d, c)$ with a bend above the Y-line. We draw the edge (d, b) with a bend above the Y-line and show that one edge within triangle $\{b, d, a\}$ cannot be drawn with only one bend (since $\{b, c, d\}$ and $\{b, d, a\}$ are symmetric, we could show the same for $\{b, c, d\}$ if we would draw edge (d, b) with no bend or a bend below the Y-line).

Now we want to draw the edges from d to e, f and h. Since we do not want to change the embedding and edge (d, b) bends above the Y-line, these edges must also bend above the Y-line and the x-coordinates of their end points must obey the order $x_b < x_h < x_f < x_e < x_c$. Next we draw the edges from a to e, f and g. Since (d, e) bends above the Y-line the edges (a, f) and (a, g) must bend below the Y-line. The order of the coordinates is nearly fixed by now: $x_d < x_b < x_g, x_h < x_f < x_e < x_a$. Now we are at a point where we cannot draw edge (b, f) without letting it cross the Y-line, since (d, h) and (a, g) have their bends in opposite directions. See Figure 7.

5 The NP-Completeness Result

In this section we prove

Theorem 3. *Given any plane graph G with n vertices and n points on a line. The mapping problem of the vertices at the points so that the edges are drawn planar and with at most one bend each is NP-complete.*

Proof: The external-hamiltonian-cycle problem for plane graphs is NP-complete since it can be used to solve the hamiltonian-cycle problem for plane graphs by an iteration over all faces of the embedding.

We call a plane graph $G = (V, E)$ (external) hamiltonian-extensible if some edges E' can be inserted without destroying planarity enabling $G' = (V, E \cup E')$ to become (external) hamiltonian. The problem as to whether a given planar graph G can be made (external) hamiltonian by inserting at most $k \geq 0$ edges is clearly NP-complete since its variant with $k = 0$ is equivalent to the (external) hamiltonian-cycle problem for planar graphs.

The equivalence of the external-hamiltonian-cycle problem and the 1-bend drawability can be seen as follows: As in the lower bound example we take n points with the same y-coordinate. If G is external-hamiltonian extensible, we take the cycle C and apply the basic technique to achieve 1-bend drawings. On the other hand, if G has a mapping on the points of the horizontal line L such that a 1-bend drawing $D(G)$ exists, there is clearly no edge which crosses line L. Otherwise, it would bend twice. Hence we can easily extend G by edges between any point p_i and p_{i+1} for $i = 1, \ldots, i-1$ such that this extension induces a hamiltonian path. The last (external) edge between p_n and p_1 can also be inserted if it does not already exist.

6 Discussion

One might argue that we are cheating regarding the lower bound example since all points with the same y-coordinate contradict the commonly used assumption of a general position of the points. On the other hand, the scenario seems quite realistic. If the objects (vertices) are required to be arranged in linear order horizontally or vertically, we get exactly the given set of points which we have already proved to be hard. Open problems:
1. Improve the area bounds, especially for the general case.
2. Extend the lower bound proof and the NP-completeness result to a set of points in general position.
3. Note that the complexity of the no-bend variant is still open, although NP-completeness is also conjectured [1].

References

1. Bose, P., On Embedding an Outer-Planar Graph in a Point Set, in: G. DiBattista (ed.), *Proc. 5th Intern. Symposium on Graph Drawing* (GD'97), LNCS 1353, Springer 1998, pp. 25-36.
2. Bose, P., M. McAllister, and J. Snoeyink. Optimal algorithms to embed trees in a point set. *Journal of Graph Algorithms and Applications* 1998.
3. Castaneda, N. and J. Urrutia., Straight line embeddings of planar graphs on point sets. *Proc. 8th Canadian conf. on Comp. Geom.*, pp. 312-318, 1996.
4. Chiba, N., and T. Nishizeki, Arboricity and subgraph listing algorithms, *SIAM J. Comput.* 14 (1985), pp. 210–223.
5. Chiba, N., and T. Nishizeki, The hamiltonian cycle problem is linear-time solvable for 4-connected planar graphs, *J. Algorithms* 10 (1989), pp. 189-211.
6. Di Battista, G., Eades, P., and R. Tamassia, I.G. Tollis, *Algorithms for Automatic Graph Drawing: An Annotated Bibliography*, Brown Univ., Tech. Rep. , 1993.
7. Garey, M.R., D.S. Johnson, The Rectilinear Steiner Tree Problem is NP-complete, *SIAM J. Appl. Math.* **32** (1977), pp. 826-834.
8. Garey, M.R., D.S. Johnson and L. Stockmeyer, Some simplified NP-complete graph problems, *Theoret. Comp. Science* 1 (1976), pp. 237–267.
9. Gritzmann, P., Mohar, B., Pach, J. and Pollack, R. Embedding a planar triangulation with vertices at specified points. in: *American Mathematical Monthly 98* (1991), pp. 165-166. (Solution to problem E3341).
10. Ikebe Y., M. Perles, A. Tamura and S. Tokunaga, The rooted tree embedding problem into points in the plane, *Discr. and Comp. Geometry* 11 (1994), pp. 51–63.
11. Lengauer, Th., *Combinatorial Algorithms for Integrated Circuit Layout*, Wiley, 1990.
12. Pach J. and R. Wenger, Embedding Planar Graphs at Fixed Vertex Locations, in: Sue Whitesides (ed.), *Proc. 6th Intern. Symposium on Graph Drawing* (GD'98), LNCS 1547, Springer 1999, pp. 263-274.
13. Pach J. and J. Töröcsik, Layout of rooted trees, in: W.T. Trotter (ed.), *Planar Graphs*, vol. 9 of DIMACS Series, pages 131-137, 1993.

The Constrained Crossing Minimization Problem

Petra Mutzel and Thomas Ziegler*

Max-Planck-Institut für Informatik, Saarbrücken,
{mutzel,tziegler}@mpi-sb.mpg.de

Abstract. In this paper we consider the *constrained crossing minimization problem* defined as follows. Given a connected planar graph $G = (V, E)$, a combinatorial embedding $\Pi(G)$ of G, and a set of pairwise distinct edges $F \subseteq V \times V$, find a drawing of $G' = (V, E \cup F)$ such that the combinatorial embedding $\Pi(G)$ of G is preserved and the number of edge crossings is minimized. The constrained crossing minimization problem arises in the graph drawing method based on planarization.
In [4] we have shown that we can formulate the constrained crossing minimization problem as an $|F|$–pairs shortest walks problem, where we want to minimize the sum of the lengths of the walks plus the number of crossings between the walks.
Here we present an integer linear programming formulation (ILP) for the *shortest crossing walks problem*. Furthermore, we will present additional valid inequalities that strengthen the formulation. Based on our results we have designed and implemented a branch and cut algorithm. Our computational experiments for the constrained crossing minimization problem on a benchmark set of graphs ([1]) are encouraging. This is the first time that practical instances of the constrained crossing minimization problem can be solved to provable optimality.

1 Introduction

The constrained crossing minimization problem arises in the area of graph drawing. In graph drawing a small number of edge crossings is one of the most important aesthetic criteria. Unfortunately, the problem of minimizing the number of crossings in a drawing is NP–hard ([2]) and so far no practically efficient algorithm exists, even for small nontrivial graphs.

A good method to draw nonplanar graphs with a small number of crossings is the planarization method. This method proceeds as follows:

(1) Compute a maximum (or maximal) planar subgraph P of the given graph.
(2) Determine a combinatorial embedding $\Pi(P)$ of P. A *combinatorial embedding* $\Pi(G)$ of a planar graph $G = (V, E)$ is an equivalence class of the planar drawings of G. It fixes for each vertex $v \in V$ the order of the incident edges in a planar drawing of G.

* Research of second author supported by ZFE, Siemens AG, Munich, Germany.

J. Kratochvíl (Ed.): GD'99, LNCS 1731, pp. 175–185, 1999.

(3) Reinsert the edge set F removed in step (1) into the fixed combinatorial embedding $\Pi(P)$ of P such that the number of crossings is minimized.
(4) Replace the crossings by new artificial vertices, draw the resulting planar graph using a planar graph drawing algorithm, and finally, replace the artificial vertices by crossings again.

Here, we investigate the problem arising in step (3), the reinsertion of the edges into the planar subgraph P preserving the combinatorial embedding $\Pi(P)$. This problem can be formulated as the constrained crossing minimization problem defined above.

We can show that the constrained crossing minimization problem is NP-hard. Our proof is based on the NP–completeness of FIXED LINEAR CROSSING NUMBER, shown in [3]. In practice it is attacked by iterative heuristics using the following observation: If only one edge needs to be inserted, the problem can be solved optimally in polynomial time by computing a shortest path in the combinatorial dual graph extended by some vertices and edges. The heuristics iteratively insert the edges using this dual graph approach. However, the result is not always acceptable and we think that the exact solution of the constrained crossing minimization problem will lead to much nicer drawings.

In [4] we have presented the first step towards an algorithm for solving practical instances of the constrained crossing minimization problem to provable optimality. We have shown that we can formulate the constrained crossing minimization problem as a shortest crossing walks problem, which is of rather combinatorial than geometric nature.

A *walk* in a graph G is an alternating sequence of vertices and edges of G, beginning and ending with a vertex, in which each edge is incident to the two vertices immediately preceding and succeeding it. We denote a walk W between two vertices v_0 and v_l by $W = v_0 e_1 v_1 \dots e_l v_l$. If all the edges of a walk are distinct, we call the walk a *trail*. If all the vertices are distinct, we call the walk a *path*.

Since it is much easier to express a set of paths than a set of walks in terms of linear inequalities, we have investigated the shortest crossing paths problem, which is essentially the shortest crossing walks problem in which the set of walks is replaced by a set of paths (see [4]). However, an optimal solution to the shortest crossing paths problem is, in general, not an optimal solution of the constrained crossing minimization problem (see Section 2).

Here we investigate the shortest crossing walks problem. The basic idea for representing a set of walks in terms of linear inequalities is to use variables for pairs of adjacent edges instead of variables for the edges only. Besides the linear inequalities needed in the ILP, we present additional classes of valid inequalities for the problem. We can solve the separation problem for most of the described inequalities. Hence, we can make use of them within a branch and cut algorithm. Our computational experiments show that our new approach is not only of theoretical but also of practical interest.

This paper is organized as follows. In Section 2 we briefly review the relation between the constrained crossing minimization problem and the shortest

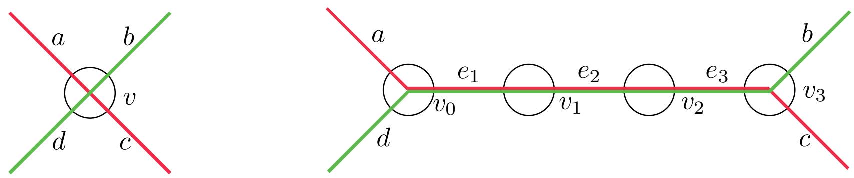


Fig. 1. On the left you see a simple crossing at vertex $v \in V$, on the right you see a distributed crossing with common subsequence $v_0e_1v_1e_2v_2e_3v_3$.

crossing walks problem. In Section 3 we present an ILP for the description of all trails between two distinct vertices of a given graph. We use this ILP in Section 4 to describe an ILP for the shortest crossing trails problem. In this section we also present additional valid inequalities for the shortest crossing trails problem. In Section 5 we apply the ILP of Section 4 to the shortest crossing walks problem. In Section 6 we briefly describe a branch and cut algorithm for the constrained crossing minimization problem and present computational results on a benchmark set of graphs [1].

2 Constrained Crossing Minimization and Shortest Crossing Walks

In this section we briefly review the relation between the constrained crossing minimization problem and the shortest crossing walks problem. More details and a proof of the equivalence can be found in [4]. For the rest of the paper we will then concentrate on the shortest crossing walks problem.

In this paper we consider undirected graphs $G = (V, E)$, which may contain loops and multiple edges. We denote an edge of the graph by $e = uv \in E$, where the order of the two end vertices is arbitrary. By $\delta(v) \subseteq E$ we denote the set of edges incident to vertex $v \in V$. We will use the notation $a - b - c$ *in* $\delta(v)$ to indicate that the edges a, b, c are pairwise distinct, incident to v, and appear in this order in the embedding $\Pi(G)$ around v.

We define the *shortest crossing walks problem* as follows. Given a weighted planar connected graph $G = (V, E)$ with embedding $\Pi(G)$ and a set $F \subseteq V \times V$ of distinct pairs of vertices of G, called *commodities*, find a set of walks in G with the following properties. There is one walk between s_k and t_k for each commodity $k = (s_k, t_k) \in F$, no walk uses an end vertex of a commodity as an internal vertex, and the sum of the weighted lengths of the walks plus the number of crossings between walks is minimum.

Figure 1 illustrates the two kinds of crossings between walks that can appear, called *simple* and *distributed crossings*. In a simple crossing two walks cross at one vertex and in a distributed crossing they have a common subsequence.

The *combinatorial dual graph* $G^\star = (V^\star, E^\star)$ of a planar graph $G = (V, E)$ with embedding $\Pi(G)$ has a vertex $v^\star \in V^\star$ for each face f of $\Pi(G)$. Corresponding to each edge $e \in E$ there is an edge $e^\star \in E^\star$ connecting the two vertices $v^\star$

and $w^\star$ corresponding to the faces incident to e. The order of the edges around a vertex $v^\star$ in $\Pi(G^\star)$ is defined by the order of the edges in G on the boundary of the face corresponding to $v^\star$. This defines the embedding $\Pi(G^\star)$ of $G^\star$ uniquely.

Given an instance, $(G, \Pi(G), F)$, of the constrained crossing minimization problem we compute the corresponding instance, $(G^\star, \Pi(G^\star), F^\star)$, of the shortest crossing walks problem as follows.

Let $G^\star$ be the combinatorial dual graph of G. We call the vertices (edges) of the combinatorial dual graph *regular vertices (edges)*. For every vertex of G that is an end vertex of an edge in F we add a new vertex in the appropriate face of $\Pi(G^\star)$. We connect each additional vertex to all vertices on the boundary of the face it was placed in (see Figure 2). The resulting graph $G^\star$ is still planar and $\Pi(G^\star)$ is uniquely determined by $\Pi(G)$. We call the resulting graph $G^\star$ the *extended dual graph*. The set $F^\star$ is the set of pairs of additional vertices corresponding to the end vertices of the edges in F. To simplify our description we replace every loop in $G^\star$ by a path of length two.

We denote the set of regular vertices (edges) by $V_R^\star \subseteq V^\star$ $(E_R^\star \subseteq E^\star)$. For each commodity $k = (s_k, t_k) \in F^\star$ we denote by $E_k^\star \subseteq E^\star$ the union of the regular edges and the edges incident to s_k and t_k, $E_k^\star = E_R^\star \cup \delta(s_k) \cup \delta(t_k)$. $E_k^\star$ is the set of edges commodity k is allowed to use. By $\delta^k(v) = E_k^\star \cap \delta(v)$ we denote the set of edges in $E_k^\star$ incident to vertex $v \in V^\star$.

We associate weight 0 with the additional edges, weight $1/2$ with the edges replacing a loop, and weight 1 with all the other edges. Using these weights on the edges the value of a solution of the shortest crossing walks problem is the same as the number of crossings in the corresponding solution of the constrained crossing minimization problem.

Figure 2(a) shows an instance of the constrained crossing minimization problem. Here G is the graph induced by the solid edges and we want to insert the dashed edges with a minimum number of crossings. Figure 2(b) shows G together with its extended dual graph. The vertices of the combinatorial dual graph are drawn as rectangles and the edges as grey solid lines. The additional vertices are $1, 3, 4, 5, 6, 7$ and the additional edges are drawn as dashed lines.

This example also shows that considering only paths in the extended dual graph is not sufficient to find an optimal solution of the constrained crossing minimization problem. The optimal solution for this example is the one shown in Figure 2(a). This solution has two crossings. However, there is no path in the extended dual graph corresponding to the edge between 3 and 7. We have to use vertex a twice. The best solution using only paths in the extended dual graph needs three crossings.

3 An ILP for Trails

In this section we present an integer linear program (ILP) describing the set of all trails in a graph $G = (V, E)$ between two distinct vertices $s, t \in V, s \neq t$.

We assume that there is no edge between s and t. Moreover, we do not use s and t as internal vertices of the trail. They only appear as first, respectively last,

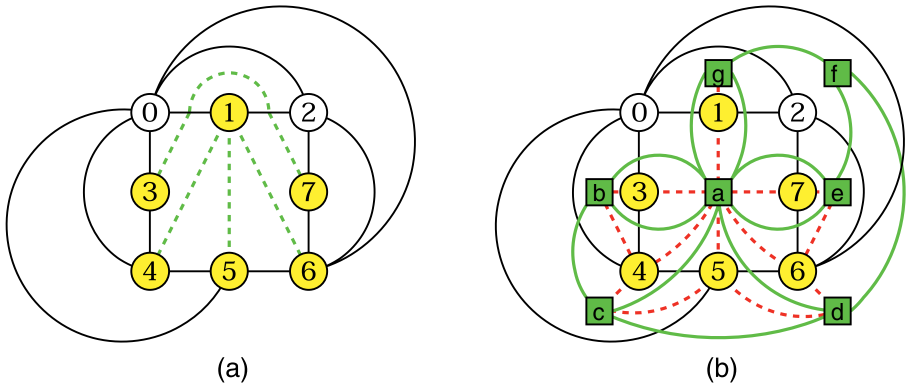


Fig. 2. Example of a graph and its extended dual graph.

vertex in the trail. These assumptions are naturally satisfied for our shortest crossing walks problem. We also assume that our graph has no loops.

To describe the trails between s and t we introduce a binary variable y^v_{ef} for every pair of adjacent edges $e, f \in \delta(v), v \in V \setminus \{s,t\}, e \neq f$, with

$$y^v_{ef} = \begin{cases} 1 \text{ , if } \mathit{evf} \text{ or } \mathit{fve} \text{ appears in the trail,} \\ 0 \text{ , otherwise.} \end{cases}$$

We say that a variable y^v_{ef} *covers* the edges e and f. We need the index v, if e and f are parallel, i.e., they have both end vertices in common.

Every trail starts in vertex s and does not meet s again. Therefore each trail uses exactly one edge incident to vertex s. For the same reason each trail uses exactly one edge incident to t. We can express these properties by

$$\sum_{e=sv\in E} \sum_{f\in\delta(v)\setminus\delta(s)} y^v_{ef} = 1 \text{ and } \sum_{e=tv\in E} \sum_{f\in\delta(v)\setminus\delta(t)} y^v_{ef} = 1. \tag{1}$$

To ensure that each edge of the graph can be used at most once, we use

$$0 \leq \sum_{f\in\delta(v)\setminus\{e\}} y^v_{ef} \leq 1 \quad \forall e = uv \in E; u, v \in V \setminus \{s,t\}. \tag{2}$$

If an edge $e = uv \in E$, $u, v \in V \setminus \{s,t\}$, is used by a trail, then there is also an edge preceding e and an edge succeeding e in the trail, that is, e has exactly one neighbor $f \in \delta(u) \setminus \{e\}$ at u in the trail and exactly one neighbor $g \in \delta(v) \setminus \{e\}$ at v. To guarantee this property we have

$$\sum_{f\in\delta(u)\setminus\{e\}} y^u_{ef} = \sum_{f\in\delta(v)\setminus\{e\}} y^v_{ef} \quad \forall e = uv \in E; u, v \in V \setminus \{s,t\}. \tag{3}$$

To complete our ILP we also need integrality constraints on the variables

$$y^v_{ef} \in \{0,1\} \quad \forall e, f \in \delta(v), v \in V \setminus \{s,t\}, e \neq f. \tag{4}$$

The correctness of the ILP can be seen as follows. A trail $T = se_1v_1 \dots e_lt$ between s and t defines in a natural way the values for the variables. We assign value one to the variables $y^{v_i}_{e_ie_{i+1}}$, for $i = 1, \dots, l-1$, corresponding to subsequences $e_iv_ie_{i+1}$ in T, and value zero to all other variables. It is not difficult to check that these variables satisfy all constraints.

Given a feasible integral solution for the ILP, we can determine the corresponding trail as follows. By (1), we have exactly one variable y^v_{ef} with value one among all variables covering an edge $e = sv \in E$ incident to s and $f = vw \in \delta(v) \setminus \delta(s)$. This defines the first and the second edge of the trail, e and f, respectively. By (2) and (3) there is exactly one variable y^w_{fg} with value one for all $g \in \delta(w) \setminus \{f\}$. This defines the third edge g of the trail. Using the variables with value one in this way, we can always find the "next edge" of the trail. This process can only stop at t. Notice however, that feasible solutions can contain additional cycles not reachable from vertex s or t.

Notice also that any feasible solution of the ILP describes a trail between s and t uniquely. We can, in particular, determine locally for each vertex the pairs of incident edges, which are used successively by the trail. This is an important property for our description of crossings between trails.

4 An ILP for the Shortest Crossing Trails Problem

We use the following ILP to describe an instance, $(G, \Pi(G), F)$, of the shortest crossing trails problem.

To describe the trails for commodity $k = (s_k, t_k) \in F$, we define for every pair of adjacent edges $e, f \in \delta^k(v), v \in V_R, e \neq f$, a binary variable $y^{vk}_{ef} \in \{0, 1\}$ and use the constraints of Section 3 on these variables.

To describe crossings between trails we introduce a binary variable z^{kl}_v for every pair of commodities $k, l \in F, k \neq l$, and every regular vertex $v \in V_R$ with

$$z^{kl}_v = \begin{cases} 1 \text{ , if trails for } k \text{ and } l \text{ cross at vertex } v, \\ 0 \text{ , otherwise.} \end{cases}$$

Let $k, l \in F, k \neq l$, be two commodities. To model simple crossings we introduce at every regular vertex $v \in V_R$ the following constraints

$$z^{kl}_v \geq y^{vk}_{ac} + y^{vl}_{bd} - 1 \quad \forall\, a, c \in \delta^k(v); b, d \in \delta^l(v); a - b - c - d \text{ in } \delta(v).$$

Such an inequality models the particular simple crossing between commodity k and l at vertex v, where the trail for commodity k uses a and c successively at v, and the trail for commodity l uses b and d successively at v (cf. Figure 1).

For distributed crossings we have the following constraints for every trail $T = v_0e_1v_1 \dots e_pv_p$ with $v_0, \dots, v_p \in V_R$.

$$\sum_{v_i \in T} z^{kl}_{v_i} \geq y^{v_0k}_{ae_1} + \sum_{i=1}^{p-1} y^{v_ik}_{e_ie_{i+1}} + y^{v_pk}_{e_pc} + y^{v_0l}_{de_1} + \sum_{i=1}^{p-1} y^{v_il}_{e_ie_{i+1}} + y^{v_pl}_{e_pb} - 2p - 1$$

$$\forall a \in \delta^k(v_0), c \in \delta^k(v_p), b \in \delta^l(v_p), d \in \delta^l(v_0),$$
$$e_1 - d - a \text{ in } \delta(v_0), \text{ and } e_p - b - c \text{ in } \delta(v_p).$$

As for simple crossings such an inequality models one particular distributed crossing between the commodities k and l with common subsequence T.

Some more explanation may be necessary here. Consider the distributed crossing shown in Figure 1. If we have this crossing in our solution, we want the sum of the crossing variables of the vertices v_0, v_1, v_2, v_3 to have value one. Therefore the left hand side sums over these variables. The right hand side of the inequality has at most value one (8 variables, each with value at most one). It has value one, when all variables on the right hand side have value one. In all other cases it is nonpositive and the inequality is trivially satisfied. If the right hand side has value one, we must have the distributed crossing in our solution. In this case also the left hand side must have value one to satisfy the inequality.

Our objective is to minimize the sum of the weighted lengths of the trails plus the number of crossings between trails.

The complete ILP looks as follows:

$$\min \sum_{k\in F}\sum_{v\in V_R}\sum_{e,f\in\delta^k(v),e\neq f}\frac{1}{2}(c_e+c_f)y_{ef}^{vk}+\sum_{v\in V_R}\sum_{k,l\in F,k\neq l} z_v^{kl} \tag{5}$$

s.t.

$$y_{ef}^{vk}\in\{0,1\}\quad \forall k\in F; e,f\in\delta^k(v), v\in V_R, e\neq f \tag{6}$$

$$z_v^{kl}\in\{0,1\}\quad \forall v\in V_R; k,l\in F, k\neq l \tag{7}$$

$$\sum_{e=s_k v\in E_k}\sum_{f\in\delta^k(v)\setminus\delta(s_k)} y_{ef}^{vk}=1\quad \forall k=(s_k,t_k)\in F \tag{8}$$

$$\sum_{e=t_k v\in E_k}\sum_{f\in\delta^k(v)\setminus\delta(t_k)} y_{ef}^{vk}=1\quad \forall k=(s_k,t_k)\in F \tag{9}$$

$$0\leq \sum_{f\in\delta^k(v)\setminus\{e\}} y_{ef}^{vk}\leq 1\quad \forall k\in F; e=uv\in E_k; u,v\in V_R \tag{10}$$

$$\sum_{f\in\delta^k(u)\setminus\{e\}} y_{ef}^{uk}=\sum_{f\in\delta^k(v)\setminus\{e\}} y_{ef}^{vk}\quad \forall k\in F; e=uv\in E_k; u,v\in V_R \tag{11}$$

$$\begin{gathered} z_v^{kl}\geq y_{ac}^{vk}+y_{bd}^{vl}-1\\ \forall k,l\in F, k\neq l; v\in V_R;\\ a,c\in\delta^k(v); b,d\in\delta^l(v); a-b-c-d \text{ in } \delta(v)\end{gathered} \tag{12}$$

$$\begin{gathered}\sum_{v_i\in T} z_{v_i}^{kl}\geq y_{ae_1}^{v_0k}+\sum_{i=1}^{p-1} y_{e_ie_{i+1}}^{v_ik}+y_{e_pc}^{v_pk}+y_{de_1}^{v_0l}+\sum_{i=1}^{p-1} y_{e_ie_{i+1}}^{v_il}+y_{e_pb}^{v_pl}-2p-1\\ \forall T=v_0e_1\ldots e_pv_p \text{ trail, with } v_i\in V_R, \text{ for } i=0,\ldots,p;\\ k,l\in F, k\neq l;\\ a\in\delta^k(v_0); c\in\delta^k(v_p); b\in\delta^l(v_p); d\in\delta^l(v_0);\\ e_1-d-a \text{ in } \delta(v_0); e_p-b-c \text{ in } \delta(v_p)\end{gathered} \tag{13}$$

The correctness of the ILP can be seen as follows. From the correctness of the ILP for trails we know that the variables y_{ef}^{vk} for commodity $k = (s_k, t_k) \in F$ describe a trail between the vertices s_k and t_k. No trail uses an additional vertex as an internal vertex. We need to show the correctness of the crossing constraints and of the objective function.

Consider the constraints for simple crossings. Every such constraint models one particular crossing at a vertex $v \in V_R$ between the commodities k and l. If this crossing appears in our solution, the right hand side of the constraint has value one. In all other cases the value of the right hand side is nonpositive. On the other hand, if one of the simple crossing constraints is tight, we also must have the corresponding simple crossing in our solution. The same argument can be used to show the correctness of the distributed crossing constraints.

The objective function has two parts. The first part computes the sum of the lengths of the trails and the second part computes the number of crossings between trails. By the correctness of the crossing constraints, two trails cross if and only if the sum of the corresponding crossing variables has value one. Therefore the second sum computes the correct value. To see the correctness of the first part consider one commodity $k = (s_k, t_k) \in F$. The variables y_{ef}^{vk} describe a trail from s_k to t_k. Each edge of the trail, except for the first and the last edge, is covered by exactly two variables with value one. Therefore c_e appears twice for each internal edge of the trail and we have to use the coefficient 1/2. Edges with weight $c_e = 1/2$ are always used in pairs, since one end vertex has degree two. This shows that we compute the length of each trail correctly and the objective function computes the desired value.

We present some classes of valid inequalities now. We do not need these inqualities in our ILP, but they are very useful in our branch and cut algorithm.

Cut Inequalities. Every trail between s_k and t_k must use at least one edge of every cut $C \subseteq E_k$ separating s_k from t_k. We can use this property to define a class of valid inequalities as follows.

$$\sum_{e=uv \in C, v \in V_R} \sum_{f \in \delta^k(v) \setminus \{e\}} y_{ef}^{vk} \geq 1 \quad \forall C \subseteq E_k \ s_k\text{–}t_k\text{–cut} \tag{14}$$

Additional Constraints for Simple Crossings. Let $v \in V_R$ be a regular vertex and $k, l \in F, k \neq l$, be two commodities. Let $\delta(v) = \{e_1, e_2, \ldots, e_r\}$ with order $e_1 - e_2 - \ldots - e_r$ in $\delta(v)$. Let $D_1 \cup D_2 \cup D_3 \cup D_4 = \delta(v)$ be a partition of $\delta(v)$ with the following properties:

- $D_1 \cap \delta^k(v) \neq \emptyset$, $D_3 \cap \delta^k(v) \neq \emptyset$, $D_2 \cap \delta^l(v) \neq \emptyset$, $D_4 \cap \delta^l(v) \neq \emptyset$.
- For arbitrary $d_1 \in D_1$, $d_2 \in D_2$, $d_3 \in D_3$, $d_4 \in D_4$: $d_1 - d_2 - d_3 - d_4$ in $\delta(v)$.

Then for all $e_k \in D_1 \cap \delta^k(v), e_l \in D_2 \cap \delta^l(v)$ the following inequalities are valid for the shortest crossing trails problem:

$$z_v^{kl} \geq \sum_{f \in D_3 \cap \delta^k(v)} y_{e_k f}^{vk} + \sum_{f \in D_4 \cap \delta^l(v)} y_{e_l f}^{vl} - 1 \tag{15}$$

In every integral solution at most one of the variables $y^{vk}_{e_k f}$ and at most one of the variables $y^{vl}_{e_l f}$ has value one, because all these variables cover the edge e_k, respectively e_l. If both sums have value one, then there is exactly one variable with value one in each sum and we have a simple crossing between k and l at vertex v by the choice of D_1, D_2, D_3, and D_4. Otherwise the right hand side of the inequality is nonpositive and the inequality is trivially satisfied.

Additional Constraints for Distributed Crossings. We can use the same idea to get new constraints for distributed crossings. Due to lack of space we can not describe these constraints here. (See [5] for details.)

5 The Shortest Crossing Walks Problem

We want to apply the ILP of Section 4 to the shortest crossing walks problem. The only difference is that walks can use edges of the graph more than once. The idea is to replace every regular edge by an appropriate number of parallel copies (without changing the embedding). Then each time a walk uses a regular edge in an optimal solution of the shortest crossing walks problem it can use a different copy in the new graph and we only have to consider trails.

It is not difficult to see that an optimal solution of the shortest crossing trails problem in the modified graph corresponds directly to an optimal solution of the constrained crossing minimization problem. We can use the same arguments as in our proof of the equivalence between the shortest crossing walks problem and the constrained crossing minimization problem (see [4]). The only problem is to find a bound on the number of copies needed for every regular edge.

The following lemma provides such a bound.

Lemma 1. *Each edge is used at most $r \leq \lfloor \frac{1}{2}(|F|) \rfloor$ times by a walk in an optimal solution of the shortest crossing walks problem.*

Proof. Assume that a walk W uses an edge r times. Then W is at least $2r - 2$ longer than a corresponding shortest path. In an optimal solution of the shortest crossing walks problem two walks can have at most one crossing and therefore the length of every single walk can be at most $|F| - 2$ longer than a corresponding shortest path. So we have $2r - 2 \leq |F| - 2$, that is, $r \leq \lfloor \frac{1}{2}(|F|) \rfloor$.

6 Computational Experiments and Results

Based on the ILP and the valid inequalities we developed a branch and cut algorithm. We start with the constraints (8),(9),(10), and (11) and use the other constraints as cutting planes. We can separate the constraints for simple crossings and cuts exactly in polynomial time. However, our separation of distributed crossing constraints is only a heuristic. (See [5] for more details).

We implemented the algorithm using LEDA [6] and ABACUS [7] and did computational experiments on a benchmark set of graphs [1]. For every graph we computed a planar subgraph and a combinatorial embedding of the planar

subgraph. The benchmark set contains 11529 graphs. In our experiments we considered only the 5157 graphs where we deleted between 2 and 10 edges. We allowed a maximum running time of one hour of CPU–time per instance on a Sun Enterprise 10000 with 12 GB main memory.

We aimed on computing provable optimal solutions for the shortest crossing paths, the shortest crossing trails, and the shortest crossing walks problem on the extended dual graphs of the benchmark graphs. Figure 3(a) shows the percentage of instances we could solve to provable optimality. Whereas we could solve all the instances with 6 commodities of the paths version to optimality, we could only do this for 95% of the instances of the walks version. In the case of 10 commodities, we could compute a provable optimal solutions for 56% of the instances of the paths and for 53% of the trails version, but only 10% of the walks version. The additional inequalities for crossings described in Section 4 are indeed useful in our branch and cut algorithm. Without them we could solve about 12% less instances to optimality within the given time. The average running times per instance is shown in Figure 3(b).

Figure 3(c) shows the number of crossings obtained by the iterative heuristic (described in Section 1), and the result of the branch and cut algorithm for the trails version. The number of crossings improves by about 7% on average compared to the heuristic. The maximal improvement, however, was 33%. Figure 3(c) also shows the trivial lower bound obtained by simply adding the lengths of the shortest paths for the commodities, and the lower bound obtained by our branch and cut algorithm. We also observed that the number of crossings of the paths version and the walks differ only very slightly from that of the trails version. We found 14 instances, where the shortest crossing trails problem finds a better solution than the shortest crossing paths problem, but no examples, where the shortest crossing walks solution improves upon the trails solution.

References

1. G. Di Battista, A. Garg, G. Liotta, R. Tamassia, E. Tassinari, and F. Vargiu. An experimental comparison of four graph drawing algorithms. *Comput. Geom. Theory Appl.,* 7:303-326, 1997.
2. M.R. Garey and D.S. Johnson. Crossing Number is NP–Complete. *SIAM J. Alg. Disc. Meth.*, Vol.4, No.3: 312–316, 1983.
3. S. Masuda, K. Nakajima, T. Kashiwabara, and T. Fujisawa. Crossing Minimization in Linear Embeddings of Graphs. In: *IEEE Transactions on Computers*, Vol.39, No.1: 124–127, 1990.
4. P. Mutzel and T. Ziegler. The Constrained Crossing Minimization Problem – A First Approach. In: *P.Kall, H.-J. Lüthi (Eds.): Operations Research Proceedings 1998, Springer-Verlag,* pp. 125-134, 1999.
5. P. Mutzel and T. Ziegler. The Constrained Crossing Minimization Problem. Forthcoming *Technical Report, MPI Informatik, Saarbrücken, Germany,* 1999.
6. K. Mehlhorn and S. Näher. LEDA: A platform for combinatorial and geometric computing. *Comm. Assoc. Comput. Mach.,* 38:96–102, 1995.
7. M. Jünger and S. Thienel. Introduction to ABACUS - A Branch-And-CUt System. *Technical Report No. 97.263, Institut für Informatik, Universität zu Köln,* 1997, to appear in *Operations Research Letters,* 1999.

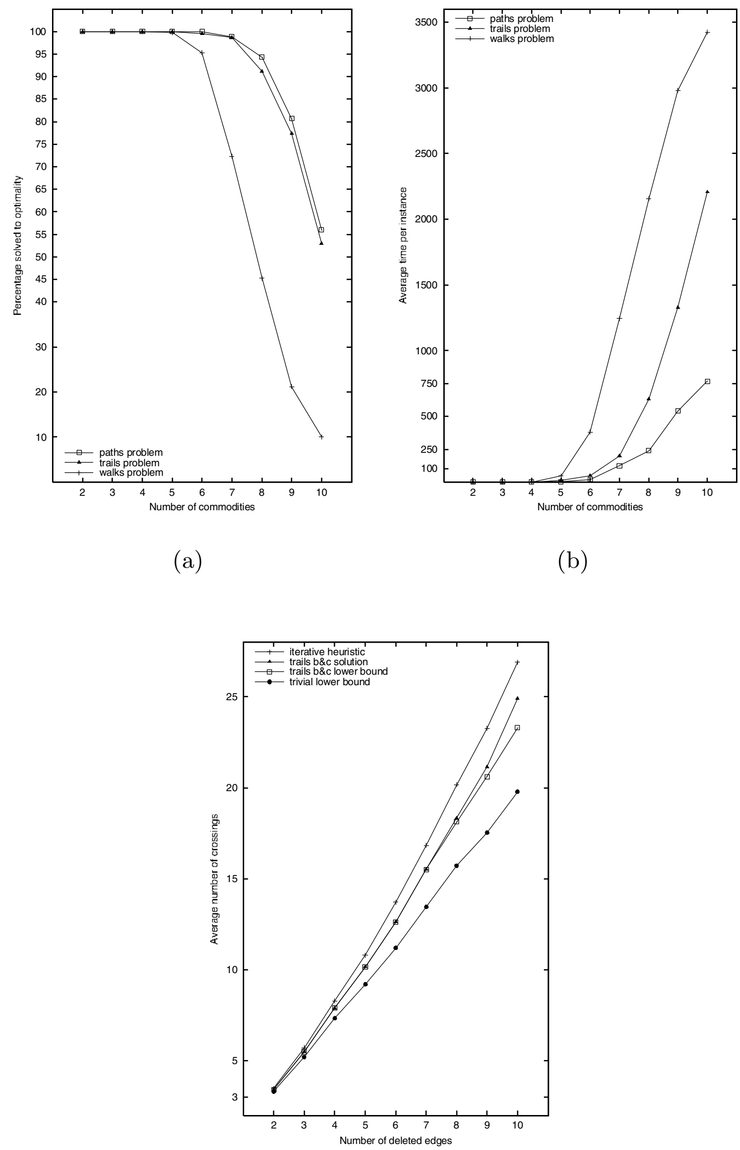


Fig. 3. Computational Results: (a) Percentage of instances solved to optimality, (b) Average running time, (c) Average number of crossings.

Planarity-Preserving Clustering and Embedding for Large Planar Graphs*

Christian A. Duncan**, Michael T. Goodrich, and Stephen G. Kobourov

Center for Geometric Computing
The Johns Hopkins University
Baltimore, MD 21218

Abstract. In this paper we present a novel approach for cluster-based drawing of large planar graphs that maintains planarity. Our technique works for arbitrary planar graphs and produces a clustering which satisfies the conditions for compound-planarity (c-planarity). Using the clustering, we obtain a representation of the graph as a collection of $O(\log n)$ layers, where each succeeding layer represents the graph in an increasing level of detail. At the same time, the difference between two graphs on neighboring layers of the hierarchy is small, thus preserving the viewer's mental map. The overall running time of the algorithm is $O(n \log n)$, where n is the number of vertices of graph G.

1 Introduction

The problem of displaying large graphs often arises in the networking and telecommunications areas. While such application areas typically give rise to non-planar graphs, there are nevertheless several application areas that give rise to large graphs that are planar. Examples of such planar graph applications include computations arising in computational cartography and geographic information systems (GIS). In this paper we are therefore concerned with the visualization of large planar graphs.

There are several approaches to the visualization of planar graphs, each of which must address the fact that the resolution of most display technologies (and possibly even the human eye) simply cannot display more than a few million pixels. Moreover, no matter how many pixels a display technology has, these pixels must display not just the vertices of a graph of interest, but also, and more importantly, the edges connecting these vertices. Our approach is based on displaying the graph using a hierarchical clustering in which the graph is represented by a collection of layers, where each succeeding layer describes the graph in a decreasing level of detail. That is, together with G one gives a tree T such that the leaves of T coincide with the vertices of G, and each internal node v of T represents the *cluster* defined by the vertices of G associated with the descendent leaves of v in T. In this case G can be drawn in a "layered" manner,

* Partially supported by NSF grant CCR-9732300 and ARO grant DAAH04-96-1-0013.
** Also at Max-Planck-Institut für Informatik.

J. Kratochvíl (Ed.): GD'99, LNCS 1731, pp. 186–196, 1999.

where we draw each cluster on the same layer of T as a region of the plane and connect adjacent clusters by segments. We would like that each such layer be drawn planar, with no segments intersecting each other or intersecting the boundary of a non-incident cluster region. Thus, the general goal of clustered graph drawing is to preserve the global structure of a graph G by recursively clustering smaller subgraphs of G and drawing these subgraphs as single nodes or filled-in regions in a rendering of G. By grouping vertices together into clusters in this way one can recursively divide a given graph into layers of decreasing detail, which can then be viewed in a top-down fashion.

1.1 Prior Related Work on Clustered Graph Drawing

If clusters of a graph are given as input along with the graph itself, then several authors give various algorithms for displaying these clusters in two or three dimensions [2,3,5,8]. Still, as will often be the case, if clusters of a graph are not given *a priori*, then various heuristics can be applied for finding clusters using properties such as connectivity, cluster size, geometric proximity, or statistical variation [7,9,11]. If no clusters are given and no special properties are known in advance, Duncan *et al.* [1] show how to create a hierarchical decomposition and a 3-dimensional drawing for general graphs. However, for planar graphs, it is possible to introduce edge-region crossings, in which edges cross cluster regions they are not part of. Even with no edge-edge crossings, the edge-region crossings are a serious drawback to the readability of a drawing.

Eades *et al.* [3] describe a drawing algorithm that draws a planar graph G, assuming that the clusters of G preserve certain recursive conditions, which they collectively call the *c-planarity* conditions. They show that if G and its clusters satisfy the c-planarity conditions, then one can produce a drawing of G such that each layer of the cluster hierarchy is drawn planar, with each vertex drawn as a convex region and each edge drawn as a straight line segment. This approach allows the graph to be represented by a sequence of drawings of increasing detail. As illustrated by Eades and Feng [2], this hierarchical approach to drawing large graphs can be very effective. However, we are not aware of any previous work for deterministically producing a clustering of an arbitrary planar graph so as to satisfy all the c-planarity conditions.

1.2 Our Results

In this paper we provide an algorithm for constructing a clustering of any planar graph so as to satisfy the c-planarity conditions of Eades *et al.* [3]. Our algorithm runs in $O(n \log n)$ time, uses $O(n)$ space, and can be implemented using simple "off-the-shelf" data structures. We also show that the clustering tree T, defined by our algorithm, has the additional property that the number of clusters at layer i of T (i.e., the clusters associated with the nodes of T at height i) is a constant fraction fewer than the number of clusters at the next higher layer, $i + 1$. Thus, T has $O(\log n)$ height. This in turn implies faster drawing times when T is used in a clustered graph drawing algorithm.

This logarithmic height result also implies some nice properties of the clustered drawing itself. For example, had we instead produced a clustering tree T of depth $\Theta(n)$, which is possible if one uses a different clustering algorithm, then we would have a hierarchy that takes an extraordinarily long time to traverse for large planar graphs. At the same time, an $o(\log n)$ height for T would imply drastic changes between consecutive layers in the hierarchy.

In addition to this logarithmic height result, our algorithm produces a clustering such that the changes between the graphs in consecutive layers of the hierarchy T are "local." In order to preserve the viewer's mental map of the graph when moving from one layer to another, the changes in the graph should be minimal. Given the graph in layer i in T, to obtain the graph associated with the next higher layer $i+1$ in T, we need to group certain sets of vertices together and replace them by new vertices. In this paper, we consider only changes that affect pairs of vertices, so that the tree T is in fact a *binary* tree.

Thus we restrict our clustering operation so as to allow only the combining of two adjacent clusters, which is an operation typically referred to as an *edge contraction*. Through a sequence of graph contractions, we obtain the layer graphs $G_0, G_1, \ldots, G_k$, where $G_0 = G$ and G_k is a singleton graph. If the changes necessary to obtain layer $i+1$ from layer i are to be local, then the following three *locality conditions for edge contraction* must be met:

1. A vertex can participate in at most one edge contraction.
2. Changes in the drawing of the graph that result from the contraction of an edge (u, v) should only affect edges with endpoint u or v.
3. A contraction of edge (u, v) results in the creation of vertex w. The placement of w in the drawing should be "close" to the edge (u, v). Optimally, we would like that w lie along the line segment defined by (u, v).

We provide a clustering method that satisfies the above locality conditions. One of the main challenges in creating the layers in a cluster hierarchy of a planar graph is to define clusters and the drawing algorithm associated with G's clustering in such a way that no edge crossings are introduced in the drawing of each layer. We provide a drawing algorithm which makes use of our clustering method to produce a drawing that has neither edge-edge crossings nor edge-region crossings. In addition, we show that one can use our clustering as input to the clustered planar graph drawing algorithm of Eades *et al.* [3].

2 Hierarchical Embedding of Planar Graphs

Let us assume, without loss of generality, that all the graphs that we are dealing with are maximally planar. If a particular graph is not maximally planar then we can fully triangulate it. Let $G = (V, E)$ be a maximally planar graph, where $|V| = n$. $V(G)$ and $E(G)$ as usual refer to the set of G's vertices and edges, respectively and the degree of a node v in graph G is $d_G(v)$. Let $l_G(f_i)$ be the length of a face f_i in G, where by the length of the face we mean the number of

vertices on that face. Further, Δuvw will refer to the triangle defined by vertices u, v, w and by the edges $(u, v), (v, w), (w, u)$.

Similar to [2] we define the *clustered graph* $C = (G, T)$ to be the graph G and a tree T such that the vertices of G coincide with the leaves of T. An internal node of T represents a cluster, which consists of all the vertices in its subtree. All the nodes of T at a given height i represent the clusters of that level. A *view at level* i, $G_i = (V(G_i), E(G_i))$, consists of the nodes of height i in T and a set of representative edges. The edge (u, v) is in $E(G_i)$ if there exists an edge between a and b in G, where a is in the subtree of u and b is in the subtree of v. Each node $u \in T$ has an associated region, corresponding to the partition given by T.

We create the graphs G_i in a bottom-up fashion, starting with $G_0 = G$ and going all the way up to G_k, where $k = \texttt{height}(T)$. We obtain G_{i+1} from G_i by contracting a carefully chosen set of edges of G_i in a certain order. The z-coordinate of a vertex $v \in V(G_i)$ is equal to i, that is, all the vertices in G_i are embedded in the plane given by $z = i$. The edges of T are defined by the edge contractions. More precisely, if $(u, v) \in E(G_i)$ is contracted to a vertex $w \in G_{i+1}$, then edges (w, u) and (w, v) are added to T.

The problem of embedding planar graphs with straight lines and no crossings is well studied [4,10,12]. Embedding clustered graphs without crossings poses additional difficulties. To embed the layers, we reverse the sequence of graph contractions: we start with embedding of G_k (which has only one vertex). To obtain an embedding for G_{i-1} from an embedding for G_i we consider the set of edges of G_{i-1} whose contraction resulted in G_i. We then reverse the process by carefully expanding and embedding one edge from that set at a time. Throughout this process we maintain the three locality conditions for edge expansions/contractions.

2.1 Edge Contraction and Separating Triangles

Contracting an edge is a standard operation on planar graphs, see [6]. We say that an edge $e = (u, v)$ of G is *contracted* when its endpoints, u and v, are replaced by a new vertex w such that all resulting multiple edges are removed. Ideally, we would like to perform edge contractions in a straight-line drawing that can be continuously animated so as to preserve planarity. Furthermore, so as to preserve the viewer's mental map, we prefer that only the endpoints of the contracted edge move, resulting in only minimal changes in the drawing.

It is well-known that contracting an edge in a planar graph results in a planar graph [6]. Note that this does not imply that contracting an edge in a straight line planar drawing of a graph results in a straight line planar drawing! More precisely, consider a straight-line planar drawing of a graph and an edge to be contracted. Suppose we are not allowed to move any other vertices in the drawing except the two involved. Then there exist drawings in which the contraction of some such edge introduces a crossing. We show this with an example in Fig. 1.

We have seen one of the problems that occur when an edge in an embedded graph is contracted. Another problem can occur even if we do not have a fixed embedding. When the contracted edge is a part of a separating triangle, the

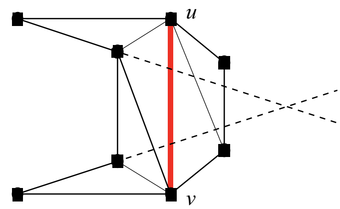


Fig. 1. A subgraph of an embedded fully triangulated graph. Edge (u, v) cannot be contracted without introducing a crossing, if we are to keep all other vertices fixed.

resulting graph is not fully triangulated and in fact may have many different embeddings. We call a triangle in G is a *separating triangle* if the removal of its vertices and their adjacent edges disconnects G.

Thus, we can divide the edges of G into two categories depending on the effect their contraction has on the resulting graph. We say that an edge is *simple* if it is not a part of a separating triangle. Edges that are part of separating triangles we call *non-simple*. Non-simple edges present problems when contracted, so we will be contracting only simple edges, for their contraction can be continuously animated while preserving planarity using straight lines. Moreover, eliminating the parallel edges after contracting a simple edge in a maximally planar graph results in a maximally planar graph.

3 Simple Matching in Maximally Planar Graphs

In this section we show that any maximal matching that uses only simple edges contains a constant fraction of all the edges in G, provided G is maximally planar. Next we show how to find a matching that can be used to contract the graph so that the resulting graph is maximally planar. Furthermore, if the size of that matching is $O(n)$, then after repeating this process $O(\log n)$ times we are left with only a constant number of vertices. Thus, we need to show that we can construct a maximal matching with $O(n)$ edges such that their contraction results in a maximally planar graph.

Let G' be the graph obtained from G by removing all the non-simple edges. We start by showing that any maximal matching in G' contains at least $n/12$ edges. To prove this claim we construct a maximal matching in G' and consider faces of different lengths. Recall that the length of a face refers to the number of vertices on that face. We break the faces of G' into three classes, A, B, C, respectively faces of length 3, faces of length 4, and faces of length 5 or more. We then count the number of unmatched nodes in faces of the different classes. Finally, when we factor in over-counting we show that any maximal matching must contain at least $n/12$ edges.

Lemma 1. *If f_i is a face in A, there is at most one unmatched vertex in f_i and this vertex has degree at least 3. If f_i is a face in B or C, then there exist at most $l(f_i)/2$ unmatched nodes in f_i.*

For the unmatched vertices on faces in A we can show that they have degree at least 3 in G'. This is not necessarily true for the unmatched vertices on faces in B or C. We show, however, that if a pair of unmatched vertices on a face in B have degrees 2 in G' then G belongs to a special class of graphs $\mathcal{H}$. If $G \notin \mathcal{H}$ then for any face $f_i \in B$, at most one vertex on that face has degree 2.

Lemma 2. *Let $\mathcal{H}$ be the class of maximally planar graphs in which there exist two vertices, u, v such that every other vertex in the graph is adjacent to both u and v. Then if $H \in \mathcal{H}$, any maximal matching that uses only simple edges contains $n/12$ or more edges.*

Lemma 3. *Let G and G' be a planar graph and the induced graph on G in which all the non-simple edges have been removed. If there exists a face in B with more than one vertex of degree 2, then $G \in \mathcal{H}$.*

We have shown that if G' has a face of length four with more than two nodes of degree 2, then $G \in \mathcal{H}$ and hence any maximal matching in G' contains at least $n/12$ edges (from Lemma 2). Finally we show that the same result holds for all maximally planar graphs.

Theorem 1. *Let G be a maximally planar graph and let M be the set of matched vertices in a maximal matching which uses only simple edges. Then $|M| \geq n/6$, where n is the number of vertices of G.*

4 Algorithm and Analysis

Before we can consider a particular embedding we must show how to obtain all the graphs in the hierarchy, $G_0, G_1, \ldots, G_k$. Recall that $G_0 = G$ is a fully triangulated planar graph on n vertices. To construct G_{i+1} from G_i we find a matching E_i of G_i and perform the graph contraction using the edges in E_i. We repeat this process until G_{i+1} is a singleton graph.

Set E_i for $0 \leq i < k$ contains a maximal matching on the edges of G_i with some added constraints. It is important that after the contraction of the edges in E_i the resulting graph G_{i+1} remains fully triangulated. In order to preserve the mental map, the three locality conditions must be maintained. Finally, in order to maintain a small hierarchical height, $|E_i|$ must be a constant fraction of the edges in G_i. Thus, the constraints that we have on E_i are as follows:

1. E_i is a matching of simple edges.
2. Using the locality conditions, contracting E_i yields a maximally planar G_{i+1}.
3. $|E_i| \geq |V(G_i)|/c$, for some constant $c > 1$.

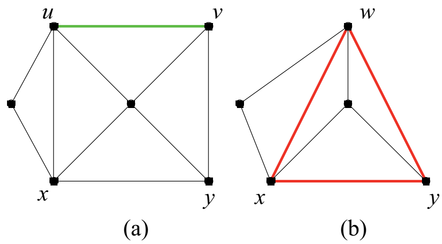


Fig. 2. Edges (u, v) and (x, y) in part (a) are both simple, do not share an endpoint, and can be contracted as a part of a matching. After (u, v) is contracted to w in part (b), edge (x, y) becomes a part of a separating triangle and so it should not be contracted.

Note that condition (1) does not imply condition (2), see Fig. 2. Before we proceed we show how to produce a set E_i which satisfies the above three conditions. Suppose we have graph G_i and we want to create set E_i so that when all the edges in E_i are contracted, we get G_{i+1}. We will contract simple edges of G_i one at a time. When an edge (u, v) is contracted, it is replaced by a node w. The next time an edge is contracted, it cannot have w as an endpoint. Let W_i be the set of vertices that were created as a result of contractions in phase i. The edges that we place in E_i must be a matching, and so when a new edge is considered for contraction, it cannot have an endpoint in W. Finally, let S_i be the set of vertices of G_i of small degrees. More precisely, let $S_i = \{v \in V(G_i) : d_{G_i}(v) < 39\}$.

In general, G_i is transformed into G_{i+1} one edge contraction at a time using the edges in E_i *in the order they were chosen.* Call the intermediate graphs from G_i to G_{i+1}, $G_i = G_{i,0}, G_{i,1}, \ldots, G_{i,j} = G_{i+1}$, and consider the algorithm on Fig. 3.

Lemma 4. *Let* $|V(G_i)| = n_i$. *Then* $|E_i| \geq n_i/50$.

Proof Sketch: For G_i with more than 3 vertices, $|E_i| \geq 1$. Then consider the sequence of intermediate graphs $G_{i,0}, G_{i,1}, \ldots, G_{i,j}$ and let $G_{i,j}$ have no more edges that could be added to E_i. Observe that we have contracted exactly j edges of G_i and so $|V(G_{i,j})| = n_i - j$. Then from Theorem 1 there are $(n_i - j)/12$ edges in any maximal matching of $G_{i,j}$ which uses only simple edges. Consider such a matching M. We are not allowed to add M edges with endpoints in W_i. But since $|W_i| = j$, at most j of the edges in M can have endpoints in W. Also note that if both endpoints of a simple edge in the matching have degrees greater than or equal to 39 in G_i they cannot be added to M. Note that if there exist at most k nodes of degree greater than or equal to 39, then there are at most $k/2$ such edges. It is easy to show that $k < n_i/12$: Suppose there are k vertices of degrees 39 or more in G_i. Since G_i is fully triangulated, every vertex has degree at least 3 and since G_i is maximally planar, the sum of the degrees

```
match(G_i, E_i)
    j ← 0
    G_{i,j} ← G_i
    W_i ← ∅
    while (S_i ≠ ∅)
        Let u_j ∈ S_i, S_i = S_i \ {u_j}
        if ∃(u_j, v_j) ∈ G_{i,j}, s.t. v_j ∉ W_i and (u_j, v_j) is simple
            E_i ← E_i ∪ {(u_j, v_j)}
            Contract (u_j, v_j) to w_j to get G_{i,j+1}
            W_i ← W_i ∪ {w_j}
            j ← j + 1
    return(G_{i,j−1})
```

Fig. 3. Create G_{i+1} from G_i by contracting a sequence of edges in E_i.

is twice the sum of the edges. Then $39k + 3(n_i - k) \leq 6n_i - 12$. From this we get that $k < n_i/12$.

We stopped selecting good edges from G_i when we got to graph $G_{i,j}$ in which we could not find a simple edge to contract. The only other types of edges that might be available in $G_{i,j}$ but which we cannot take are those that were at some point non-simple, but later became simple. Also, there can be at most j such edges. Then $(n_i - j)/12 - 2j - n_i/24 \leq 0$ which implies $j \geq n_i/50$. Thus, if we cannot find another edge to add to the matching, we must have $|E_i| = j \geq n_i/50$ which completes the proof sketch. □

From the result above it follows that the height of the hierarchy is $O(\log n)$. We next argue that one call to `match`(G_i, E_i) takes $O(n_i \log n_i)$ time, and since n_{i+1} is a constant fraction of n_i, the $O(\log n)$ calls to `match`(G_i, E_i) take $O(n \log n)$ time overall thus yielding the desired theorem:

Theorem 2. *The clustering algorithm runs in $O(n \log n)$ time and produces a sequence of graphs $G_0, G_1, \ldots, G_k$ such that G_i is maximally planar for all $0 \leq i \leq k$ and $k = O(\log n)$.*

5 Constructing the Embedding

After we obtain the combinatorial graphs $G_0, G_1, \ldots, G_k$ we have to embed them in planes $z = 0, z = 1, \ldots, z = k$. While constructing the combinatorial graphs is a bottom up process, constructing the embedding is a top-down one. The first graph to be embedded is G_k, which only has one vertex. We then expand the edges in E_{k-1} one at a time, in the reverse order of their insertion. We then argue that this can be done in a way which guarantees that no crossings are introduced. We need the following lemma.

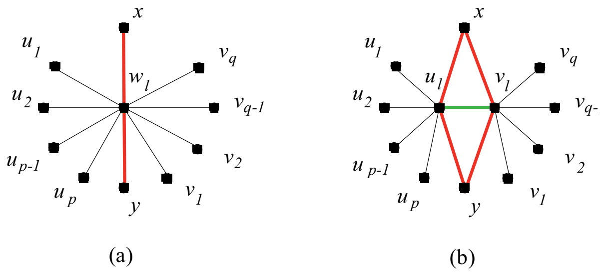


Fig. 4. Vertex w_l and its neighbors in $G_{i,l+1}$ (a) before expansion (b) after expansion.

Lemma 5. *Let G be a maximally planar graph embedded in the plane without crossings. For any $v \in V(G)$, there exists a ball of radius $\epsilon > 0$ such that if v is placed anywhere inside that ball, the embedding has no crossings.*

Proof Sketch: The main idea is to consider the visibility region around vertex v. Any point inside that region can "see" all the neighbors of v. It is not hard to show that this region cannot be empty. This would imply the existence of $\epsilon > 0$ for which the ball of size ϵ fits inside the visibility region. □

Theorem 3. *Given combinatorial representations of graphs $G_k, G_{k-1}, \ldots, G_0$ we can embed them in the planes $z = k, z = k-1, \ldots, z = 0$ so that there are no crossings in any of the drawings.*

Proof Sketch: We first embed G_k in the plane $z = k$ without crossings using any straight-line drawing method. Suppose we have embedded $G_k, G_{k-1}, \ldots, G_i$. We will show how to embed G_{i-1} given an embedding for G_i. Recall that we obtained G_i from G_{i-1} through a series of edge contractions from the edge set $E_{i-1} = \{(u_0, v_0), (u_1, v_1), \ldots (uj, vj)\}$ which produced graphs $G_{i-1,0}, G_{i-1,1}, \ldots,$ $G_{i-1,j} = G_i$. We now reverse the process and expand G_i back to G_{i-1} through the exact opposite sequence of expansions. Since we have an embedding for G_i in the plane $z = i$, we can embed $G_{i-1,j}$ in the plane $z = i-1$. Next we expand edge (u_j, v_j) by replacing vertex w_j by the pair u_j, v_j. The resulting graph is $G_{i,j-1}$ and we embed it without a crossing. We proceed until we get to $G_{i,0}$. We next show how to embed $G_{i,l}$ given an embedding for $G_{i,l+1}$, for $0 \leq l < j$.

Assume we have an straight-line embedding for $G_{i,l+1}$ without crossings on the plane $z = i$. To get $G_{i,l}$ we must expand vertex w_l back to edge (u_l, v_l). Consider the subgraph on Fig. 4. Let x and y be the neighbors in common for u_l and v_l. We then consider the ball of maximal radius around w_l which sees all neighbors (we know it is of radius $\epsilon > 0$ from Lemma 5). Consider a diameter in this ball which is perpendicular to the line connecting x and y. Place u_l and v_l on the two ends of the diagonal. □

We define the *drawing of a clustered graph* $C = (G, T)$ as in [3]. Graph G is drawn as usual, while for every node $v \in T$ the cluster is drawn as a simple closed region R such that:

– all sub-cluster regions of R are completely contained in the interior of R.
– all other cluster regions are completely contained in the exterior of R.
– if there is an edge e between two vertices contained in a cluster ν, then the drawing of e is completely contained in R.

Following the definitions of Eades *et al.*, the drawing of edge e and region R have an *edge crossing* if the drawing of e crosses the boundary of R more than once. A drawing of a clustered graph is *c-planar* if there are no edge crossings or edge-region crossings. Graphs with c-planar drawings are c-planar.

Theorem 4. *The clustered graph $C = (G, T)$ produced by our algorithm is c-planar and a c-planar embedding can be obtained in $O(n^2)$ time.*

Proof Sketch: It suffices to show that there exists a drawing of C which has no edge crossings and no edge-region crossings. Let us embed G using any planar embedding algorithm. Define the region of a cluster, ν to be the simple closed curve around the subgraph of G induced by the cluster, $G(\nu)$. By the definition of the clustering in our algorithm, the subgraph $G(\nu)$ is connected.

If u is a vertex not in cluster ν, then u cannot be contained inside the region R. Assume that u is contained in R. If we contract the edges of ν in the order defined by our algorithm, eventually u will be inside a triangular face. But then none of the edges on that face can be contracted. This is a contradiction since ν is eventually contracted to one vertex.

Finally, since G is embedded in the plane without crossings and the regions are connected there can be neither edge crossings nor edge-region crossings. Therefore C is c-planar and from [3] it follows that the c-planar embedding can be produced in $O(n^2)$ time. □

References

1. C. A. Duncan, M. T. Goodrich, and S. G. Kobourov. Balanced aspect ratio trees and their use for drawing very large graphs. *Proc. of 6th Symposium on Graph Drawing (GD'98)*, LNCS 1190:101–112, 1998.
2. P. Eades and Q. W. Feng. Multilevel visualization of clustered graphs. *Proc. of 4th Symposium on Graph Drawing (GD'96)*, LNCS 1190:101–112, 1996.
3. P. Eades, Q. W. Feng, and X. Lin. Straight-line drawing algorithms for hierarchical graphs and clustered graphs. *Proc. of the 4th Symposium on Graph Drawing (GD'96)*, LNCS 1190:113–128, 1997.
4. I. Fary. On straight lines representation of planar graphs. *Acta Sci. Math. Szeged*, 11:229–233, 1948.
5. Q.-W. Feng, R. F. Cohen, and P. Eades. Planarity for clustered graphs. *ESA'95*, LNCS 979:213–226, 1995.
6. R. J. Lipton and R. E. Tarjan. A separator theorem for planar graphs. *SIAM J. Appl. Math.*, 36:177–189, 1979.
7. F. J. Newbery. Edge concentration: A method for clustering directed graphs. In *Proceedings of the 2nd International Workshop on Software Configuration Management*, pages 76–85, Princeton, New Jersey, October 1989.

8. S. C. North. Drawing ranked digraphs with recursive clusters. *ALCOM International Workshop PARIS 1993 on Graph Drawing and Topological Graph Algorithms (GD'93)*, September 1993.
9. Sablowski and Frick. Automatic graph clustering. *Proc. of 4th Symposium on Graph Drawing (GD'96)*, LNCS 1190:395–400, 1996.
10. W. Schnyder. Embedding planar graphs on the grid. In *Proc. 1st ACM-SIAM Sympos. Discrete Algorithms*, pages 138–148, 1990.
11. K. Sugiyama and K. Misue. Visualization of structural information: Automatic drawing of compound digraphs. *IEEE Trans. Softw. Eng.*, 21(4):876–892, 1991.
12. K. Wagner. Bemerkungen zum vierfarbenproblem. *Jahresbericht der Deutschen Mathematiker-Vereinigung*, 46:26–32, 1936.

An Algorithm for Drawing Compound Graphs

François Bertault and Mirka Miller

Department of Computer Science and Software Engineering
University of Newcastle
Callaghan 2308 NSW Australia
{francois,mirka}@cs.newcastle.edu.au

Abstract. We present a new algorithm, called NUAGE, for drawing graphs with both adjacency and inclusion relationships between nodes, that is, compound graphs. Compound graphs are more general than classical graph models or clustered graphs. NUAGE can be applied to both directed and undirected compound graphs. It can be parameterized by classical graph drawing algorithms. NUAGE can be viewed as a method for unifying several classical algorithms within the same drawing by using the structure of the compound graph. Additionally, we present a refinement technique that can be used in conjunction with NUAGE to reduce the number of edge crossings.

1 Introduction

Most current systems for visualization of relational information are based on graphs. The objects or entities are the nodes of the graph, the relationships are edges between two nodes. The drawing of such graphs has received much attention and several algorithms dedicated to graphs with given combinatorial properties are available [1].

However, when we need to represent complicated relational information, with different kinds of relationships or with large amount of information, the classical graph model is not sufficient. Consequently, several extensions to the classical graph model have been proposed. Models using inclusion relationships between entities are particularly well suited for the representation of large amount of information because we can reduce the amount of information displayed by representing a set of nodes by the node that encompasses this set. A very general model for representing complex relationships is the higraph model [5]. Higraph model can represent inclusions, intersections and adjacency relationships but drawings algorithms for this model are difficult to design. Another commonly used model, called clustered graphs, consists of a classical graph and a recursive partition of the nodes of the graph. Clustered graphs model is well suited for graph drawing and several drawing algorithms have been presented [3,2]. An intermediate model between clustered graphs and higraphs, called compound graphs, has been introduced by Sugiyama and Misue for representing graphs with both inclusion and adjacency relationships. Algorithms for drawing compound digraphs have been proposed [8,9,6]. They are all based on an extension

J. Kratochvíl (Ed.): GD'99, LNCS 1731, pp. 197–204, 1999.

of hierarchical layered drawings of directed graphs and are ineffective for representing undirected edges. The approach taken in our algorithm, called NUAGE, is quite different and it does allow the representation of both directed and undirected compound graphs. Our algorithm can be viewed as a method for unifying several classical algorithms within the same drawing by using the structure of the compound graph.

2 Definitions

A *graph* $G = (V, E)$ is defined by a finite set V of *nodes* and a finite set E of *edges*, that is, unordered pairs (a, b) of nodes. If the pairs (a, b) are ordered then G is called a *directed graph.* If $(a, b) \in E$, a and b are said to be *adjacent* and a and b are called the *ends* of the edge. We denote $N_G^-(a) = \{b \mid (b, a) \in E\}$ and $N_G^+(a) = \{b \mid (a, b) \in E\}$. A *path* of length s between a node a_1 and a node a_s is a sequence of nodes $a_1, a_2, \ldots, a_s$ such that $(a_i, a_{i+1}) \in E$, for $i = 1, \ldots, s-1$. A *subgraph* $H = G|V'$ of a graph $G = (V, E)$, where $V' \subseteq V$, is the graph $H = (V', E')$ such that an edge $(a, b) \in E'$ if and only if $(a, b) \in E$.

A *rooted tree* $T = (V, E, r)$ is a directed graph such that for every node $a \in V$, except for the node r called the *root* of the tree, there is a unique path $Path_G(r, a)$ between r and a. For every node $a \in V$, except r, there exists a unique node $P_T(a)$, called the *parent* of a, such that $(P_T(a), a) \in E$. All the nodes on the path from the root to a node a (Except a itself) are called *ancestors* of a. For a node $a \in V$, the nodes in the set $N_T^+(a)$ are called the *children* of a. If $N_T^+(a)$ is empty, a is called a *leaf* of the tree, otherwise a is called an *internal node.* The *level* $L_T(a)$ of a node a is the length of the path between r and a, and by convention $L_T(r) = 0$.

A *compound graph* $C = (G, T)$ is defined as either a directed or an undirected graph $G = (V, E_G)$ and a rooted tree $T = (V, E_T, r)$ that share the same set of nodes, where all edges $(a, b) \in E_G$ are such that $a \notin Path_T(r, b)$ and $b \notin Path_T(r, a)$.

A *nested graph* $N = (G, T)$ is a compound graph such that:

$$\forall (a, b) \in E, P_T(a) = P_T(b). \tag{2.1}$$

Note that compound graphs, nested graphs and clustered graphs are all different models of graphs. Clustered graphs are compound graphs where edges in the graph are only between leaves of the tree. Nested graphs are compounds graphs where edges in the graph are only between children of the same parent.

3 Algorithm for Drawing Compound Graphs

In this section we present a new algorithm, called NUAGE, for the drawing of a compound graph $C = (G, T)$. We represent nodes by rectangles so that a node a is included in the rectangle that represents the node $P_T(a)$. For example, the compound graph defined by the tree and the directed graph of Fig. 1 is represented on the left of the Fig. 2. There is a 1-1 correspondence between the structure of the tree and the set of inclusions between nodes.

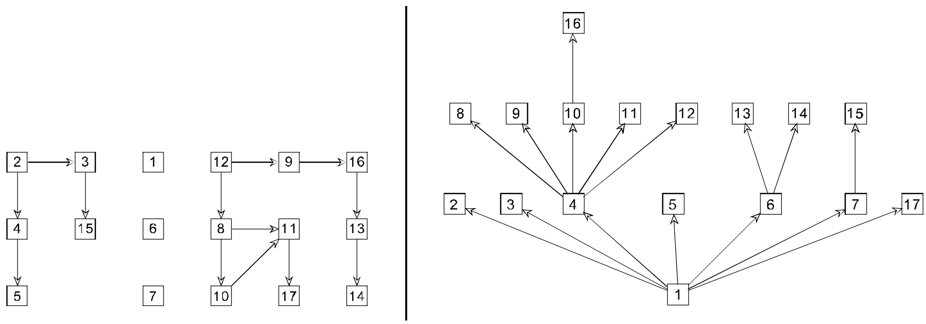


Fig. 1. Compound graph defined by a digraph and a tree.

3.1 Description of the NUAGE Algorithm

NUAGE is based on the construction of a nested graph that shares the nodes of the compound graph. The position of the nodes of the compound graph is obtained by applying an algorithm for drawing nested graphs. The construction of the nested graph is made so that the drawing of the nested graph reflects the information contained in the compound graph.

We consider a compound graph $C = (G, T)$, with $G = (V, E_G)$ and $T = (V, E_T, root)$. The steps of the NUAGE algorithm are:

Step 1. Build a nested graph $N = (H, T)$, with $H = (V, E_H)$: if $e = (a, b) \in E_g$ and $P_T(a) = P_T(b)$ then $e \in E_H$. Otherwise, e is replaced by $e' = (a', b') \in E_H$ such that a' is an ancestor of a and b' is an ancestor of b with $a' \neq b'$ and $P_T(a') = P_T(b')$.

Step 2. Apply to H an algorithm for drawing nested graphs.

3.2 Step 1: Construction of a Nested Graph Associated with a Compound Graph

The construction of the nested graph N associated with the compound graph C is based on the following principle. We start with an empty set of edges for the graph H of the nested graph. Then, for each edge (a, b) in G, we search for two distinct nodes a', an ancestor of a, and b', an ancestor of b, with $P_T(a') = P_T(b')$. We then add the edge (a', b') to H. The intuition is that if an edge influences (albeit indirectly) the position of a node v, it will also influence the position of all the nodes included in v. Thus if we replace an end v of an edge by a node which contains v, the edge will continue to have an influence on the position of v.

Algorithm (Nested graph associated with a compound graph)

- *Input:* A compound graph $C = (G, T)$, with $G = (V, E_G)$ and $T = (V, E_T, root)$.
- *Output:* A nested graph $N = (H, T)$, with $H = (V, E_H)$ and $|E_H| \leq |E_G|$.

```
N:=(H, T) with H = (V,∅);
FOR ALL (a,b)∈G DO
    a':=a; b':=b;
    WHILE P_T(a')≠P_T(b') DO
        IF L_T(a') = L_T(b') THEN
            a':=P_T(a'); b':=P_T(b');
        ELSE_IF L_T(a')>L_T(b') THEN
            a':=P_T(a');
        ELSE b':=P_T(b');
    END_WHILE;
    add edge (a',b') to H;
```

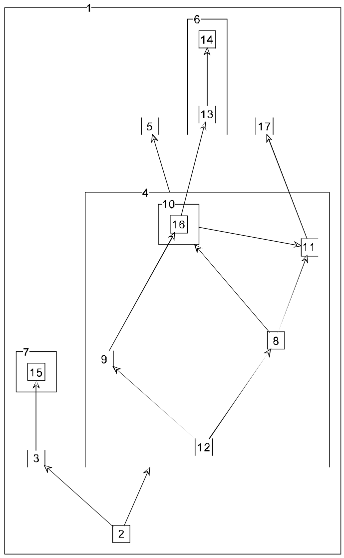
The graph on the right of figure 2 is an example of nested graph obtained after the application of the first step of the algorithm on the compound graph on the left of Fig. 2. The complexity of the first step of the algorithm Nuage is $O(|E|\log|V|)$ in the average case and $O(|E||V|)$ in the worst case. Indeed, for each edge, the number of steps of the `WHILE` loop is bounded by the height of the tree T.

3.3 Step 2: Drawing of the Nested Graph

The second step of Nuage is to draw the nested graphs associated with the compound graph. This gives us the position and the size of each node of the compound graph. The algorithm that we propose for drawing nested graphs is based on the application of classical graph drawing algorithms to each subgraph defined by the nodes with a same parent in the tree. This algorithm is called Fleur.

We consider a nested graph $N = (G,T)$, with $G = (V,E_G)$ and $T = (V,E_T,r)$. The position of a node $v \in V$ is denoted by $(x(v),y(v))$. The rectangle that represents a node $v \in V$ is denoted $r(v)$. Given a drawing of a graph G', we can define the smallest rectangle that includes the nodes and edges of G'. We call $bb(v,G')$ the operation which assigns to $r(v)$ the smallest rectangle that includes the graph G'.

We denote by $\mathcal{A}$ the class of algorithms that can be applied to undirected or directed graphs, depending on whether we consider a directed or undirected compound graph. For each internal node of the tree of a compound graph, we define a *mode* function M which takes an internal node of the tree T as an argument and returns an algorithm in $\mathcal{A}$. For example, in Fig. 2, we chose the mode of the node 4 to be a force-directed algorithm for drawing undirected graphs [4], and the mode of nodes 1 and 6 to be an algorithm for drawing trees [7]. The mode information of a node v indicates which algorithm is to be used for drawing the subgraph formed by the nodes with v as the parent in the tree T. The function that applies the algorithm $M(v)$ is called `position_`$M(v)$. We assume that the mode algorithms have the property of avoiding overlaps between the rectangles of nodes. Alternatively, we can always apply simple post-processing, after some of the mode algorithms, to remove overlaps between the rectangles of nodes.



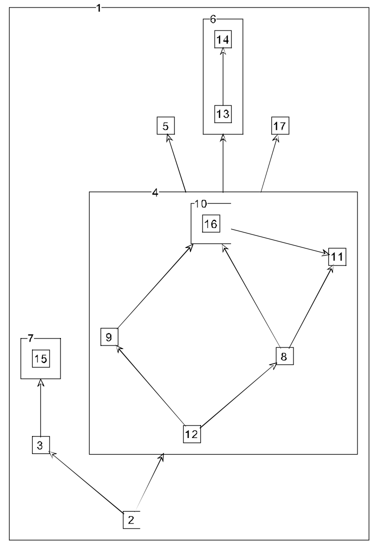


Fig. 2. Left: Drawing of the compound graph of Fig. 1. Right: Nested graph associated with the compound graph on the left.

The algorithm for drawing nested graphs is simple. We perform a depth first search traversal of the tree T and, for each internal node v of the tree, we consider the graph $H = G|_{N_T^+(v)}$, formed by the nodes with v as parent in the tree. We continue the tree traversal to calculate the width and height of the rectangles associated with the nodes in H. Then we compute the positions of the nodes in H by applying the algorithm given by $M(v)$. Next we determine the width and height of $r(v)$ by applying the $bb(v, H)$ operation. The absolute positions of the nodes in H are then set to be relative to the position of node v. After the tree traversal, we transform the relative positions of the nodes into absolute positions. The algorithm is described below.

Algorithm (Drawing of nested graphs)

- *Input:* A nested graph $N = (G, T)$, with $G = (V, E_G)$ and $T = (V, E_T, r)$. The size of the rectangle of each leaf of T is known. The mode function is defined for each internal node of T.
- *Output:* The position, width and height of each node of N is known.

```
PROCEDURE relative_position(v:node)
 IF N_T^+(v)≠∅ THEN
   FOR ALL s ∈ N_T^+(v) DO
     relative_position(s); //determine the size of rectangles of the nodes in N_T^+(v)
   H:=G|_{N_T^+(v)};
   position_M(v)(H); //we determine the positions of nodes in H
   bb(v,H); //compute the size of r(v)
   FOR ALL s ∈ N_T^+(v) DO    //the positions of nodes in N_T^+(v) are set relatively to v
     x(s):=x(s)-x(v); y(s):=y(s)-y(v);
```

```
PROCEDURE absolute_position(v:node, dx:INTEGER, dy:INTEGER)
  x(v):=x(v)+dx; y(v):=y(v)+dy;
  FOR ALL s∈N_T^+(v) DO
     absolute_position(s,x(v),y(v));

PROCEDURE FLEUR(N)
 relative_position(r); absolute_position(r,0,0);
```

The complexity of the algorithm depends on the complexity of the mode algorithms. If the overall complexity of the modes algorithms is $O(T(G))$ for a graph G, the cost of the Fleur algorithm is given by $O(|V|+\sum_{v\in V} T(G|_{N_T^+(v)}))$. If the mode algorithms are linear in the number of nodes, the Fleur algorithm remains linear.

At this point, we have completely specified the Nuage algorithm which consists of the two steps described below.

Step 1. Build the nested graph H associated with the compound graph.
Step 2. Apply to H the Fleur algorithm for drawing nested graph.

Note that if we consider one node v of the compound graph C, we can see that during the computation of the positions of the nodes $N_T^+(v)$ included in v, Nuage simply ignores all the edges between these nodes and the nodes not included in v because these edges are ignored in the nested graph. This can lead to edge crossings that could be easily removed by exchanging positions of the nodes. By intuition, we can see that these particular nodes should be placed near the boundaries of the node v. For example, if we consider the compound graph (a) in Fig. 3, we can remove an edge crossing just by swapping the positions of nodes 5 and 17. In the next section we will consider some refinement to Nuage.

4 Refinement Technique

We consider a refinement step which requires the following property of the mode algorithms. Given starting positions of the nodes, it is desirable for the mode algorithms to preserve the nodes' relative positions. For example, if we consider an algorithm that gives a vertical representation of a non-planar tree (i.e., a tree in which there is no fixed order between the children of a node), the order of the children of a node in the drawing should be according to their original horizontal coordinates.

Given a compound graph representation, obtained by applying the Nuage algorithm, the refinement step modifies the initial positions of the nodes. Since the inclusion modes algorithms should keep the relative positions of nodes, the refinement step helps to reduce the edge lengths and remove edge crossings.

To implement the refinement step, we build a new graph G_r that shares the nodes of the compound graph. The initial node positions are obtained by applying a force-directed algorithm, in which we ignore the repulsion forces. The construction of the graph G_r is very similar to the construction of the associated nested graph:

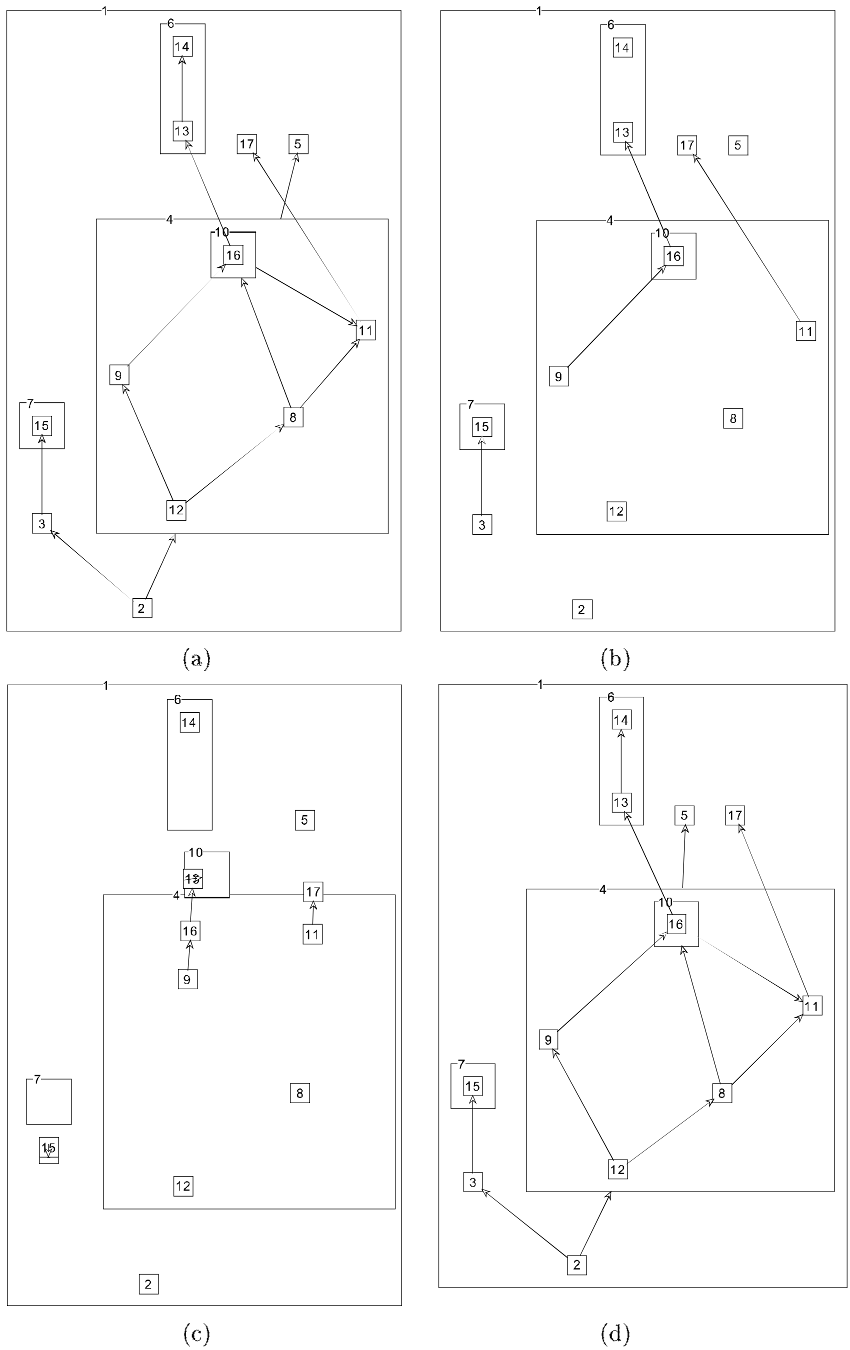


Fig. 3. Refinement step (a) the initial compound graph (b) the refinement graph associated with the compound graph (c) positions of the nodes after the contraction algorithm (d) final drawing of the compound graph.

Algorithm (Refinement graph associated with a compound graph)

- *Input:* A compound graph $C = (G, T)$, with $T = (V, E_t, r)$ and $G = (V, E_G)$.
- *Output:* The corresponding refinement graph $G_r = (V, E_r)$.

```
Gr = (V,∅);
FOR ALL (a,b)∈EG DO
    a':=a; b':=b;
    WHILE PT(a')≠PT(b') DO
        add edge (a',b') to Cr;
        IF LT(a') = LT(b') THEN
            a':=PT(a'); b':=PT(b');
        ELSE_IF LT(a')>LT(b') THEN
            a':=PT(a');
        ELSE b':=PT(b');
    END_WHILE;
```

The complexity of the refinement step is $O(|E| \log |V|)$ in the average case. Indeed, for each edge in the initial compound graph, we add $\log |V|$ edges in the refinement graph and the attraction algorithm is linear in the number of edges.

We presented an algorithm, called NUAGE, for drawing both directed and undirected compound graphs. NUAGE can be parameterized by arbitrary classical graph drawing algorithms. We implemented NUAGE in Java.

References

1. G. Di Battista, P. D. Eades, R. Tamassia, and I. G. Tollis. Algorithms for drawing graphs: an annotated bibliography,. *Computational Geometry: theory and applications*, 4(5):235–282, 1994.
2. P. D. Eades and Q.-W. Feng. Multilevel visualisation of clustered graphs. In S. North, editor, *Graph drawing*, volume 1190, pages 101–111, Berkeley (USA), 1996. Springer Verlag.
3. P. D. Eades, Q.-W. Feng, and X. Lin. Straight-line drawing algorithms for hierarchical graphs and clustered graphs. In S. North, editor, *Graph drawing*, volume 1190, pages 113–127, Berkeley (USA), 1996. Springer Verlag.
4. T. Fruchterman and E. Reingold. Graph drawing by force-directed placement. *Software-Practice and Experience*, 21(11):1129–1164, 1991.
5. D. Harel. On visual formalisms. *Communications of the ACM*, 31(5):514–530, 1988.
6. S. C. North. Drawing ranked digraphs with recursive clusters. *ALCOM Workshop on Graph Drawing 93*, 1993.
7. E. M. Reingold and J. S. Tilford. Tidier drawings of trees. *IEEE Transactions on Software Engineering*, SE-7(2):223–228, March 1981.
8. K. Sugiyama and K. Misue. Visualisation of structural information: automatic drawing of compound digraphs. *IEEE Trans. SMC*, 4(21):876–893, 1991.
9. K. Sugiyama and K. Misue. A generic compound graph visualizer/manipulator : D-Abductor. In F. J. Brandenburg, editor, *Graph Drawing*, volume 1027 of *Lecture Notes in Computer Science*, pages 500–503, Passau (Allemagne), 1995. Springer Verlag.

The Vertex-Exchange Graph: A New Concept for Multi-level Crossing Minimisation

Patrick Healy and Ago Kuusik

Department of Computer Science and Information Systems
University of Limerick
Limerick, IRELAND
{Patrick.Healy, Ago.Kuusik}@ul.ie

Abstract. In this paper we consider the problems of testing a multi-level graph for planarity and laying out a multi-level graph. We introduce a new abstraction that we call a vertex-exchange graph. We demonstrate how this concept can be used to solve these problems by providing clear and simple algorithms for testing a multi-level graph for planarity and laying out a multi-level graph when planar. We also show how the concept can be used to solve other problems relating to multi-level graph layout.

1 Introduction

Multi-level crossing minimisation is a well-known problem in graph drawing. Given a layered graph, the multi-level crossing minimisation (MLCM) problem is to reorder vertices on each level so that the number of edge crossings is minimum. The problem is NP-hard, even when there are only two layers and one layer is fixed [4].

Despite the existence of an effective heuristic solution – the Sugiyama algorithm [9] – little is known about the multi-level crossing minimisation problem itself. In particular, questions like what is a graph's crossing number (or bounds on it), what are the subgraphs that can be embedded without any crossings, or how does the ordering of one level influence edge crossings with adjacent levels, do not have answers. We believe that one of the reasons why these questions are still not answered is the lack of a theoretical mechanism giving a global view of the problem. In this paper we propose such a mechanism, the *vertex-exchange graph*. The vertex-exchange graph provides an understanding of how edge crossings relate to each other, it facilitates calculation of a level planar embedding (if it exists) and multi-level crossing minimisation, and can be used to determine successively tighter lower bounds on the crossing number. It may also provide a method for determining a tight upper bound on the number of crossings in the graph.

A multi-level (layer) graph is one where the vertices are placed on discrete levels and edges are allowed only between vertices of adjacent levels. Traditionally, multi-level graph layout algorithms have had three steps [1]. The first step creates a proper levelling of the graph, the second step finds a permutation

J. Kratochvíl (Ed.): GD'99, LNCS 1731, pp. 205–216, 1999.

of vertices on levels minimising the edge crossings, and the final step balances the layout by adjusting each vertex's x-co-ordinate. The algorithm has many refinements and modifications [1,3].

In this paper, we investigate the second step of the multi-level crossing minimisation – the permuting of vertices on levels. The most popular multi-level crossing minimisation algorithm to date is the Sugiyama algorithm [9]. The algorithm is actually a general framework which solves the multi-level crossing minimisation problem as a series of one-level crossing minimisation problems: levels are successively visited (this is called layer-by-layer sweep) and each level is ordered by some one-level crossing minimisation heuristics. Specific implementations of Sugiyama algorithm differ in three aspects: one-level crossing minimisation heuristics, sweeping directions, and stopping criterion.

The choice of the one-level crossing minimisation heuristic has a great influence on the algorithm's speed and accuracy. The barycenter [9] and median [4] heuristics have been preferred the most although there is experimental evidence [7] that the $O(n \log n)$-time *Split* heuristic [2] is superior to the two previous linear-time algorithms. In their experimental study [7] the authors also compared several one-level techniques in the *two-level* crossing minimisation. An interesting outcome was that the barycenter heuristic gave the best results and it outperformed even the authors' own Integer Linear Programming (ILP) technique. This result suggests that the multi-level crossing minimisation is less understood than one-level crossing minimisation.

On the other hand, there may exist good alternatives to the layer-by-layer sweep algorithms. One of the already existing alternatives is an ILP approach by Jünger *et al.* [6]. There exists also an evolutionary algorithmic approach by Utech *et al.*[10] that, naturally, cannot be considered to solve the crossing problem level by level. However it combines a solution to this problem with a solution to the previous step (levelling) so we do not consider it to be a solution to the crossing minimisation problem *per se*.

In the following section we introduce the vertex-exchange graph and use it to derive sufficient conditions for level-planarity of a graph. In Sections 3 and 4, respectively, we present algorithms for planarity testing of multi-level graphs and layout of planar multi-level graphs. We then show a further application by using it in an ILP formulation to find the embedding of a graph with the minimum number of crossings. Section 6 concludes the paper.

2 The Vertex-Exchange Graph

As our paper deals with graphs already having a proper levelling, we define a *proper level graph* formally.

Definition 1. *A proper level graph is a graph* $G = (V, E)$*, with vertex set* $V = V_1 \cup V_2 \cup \ldots \cup V_p$*,* $V_i \cap V_j = \emptyset$*,* $i \neq j$*, and edge set* $E = E_1 \cup E_2 \cup \ldots \cup E_{p-1}$*,* $E_i \subseteq V_i \times V_{i+1}$*.*

In what follows, when we refer to a level graph we assume that it is a proper level graph in addition.

We introduce the notation $\langle v, w \rangle$ for an *unordered* pair of same-level vertices v and w: $\langle v, w \rangle \equiv \langle w, v \rangle$. Analogously, for two edges $e, f \in E_r$, $\langle e, f \rangle$ denotes a pair of edges. In contrast, we denote an *ordered* pair of same-level vertices v and w by $[v, w]$, and $[v, w] \not\equiv [w, v]$

2.1 Definition and Properties

We now define the vertex-exchange graph of $G = (V, E)$ as follows.

Definition 2. *The vertex-exchange graph of a level graph* $G = (V, E)$ *is a graph* $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ *with vertex set* $\mathcal{V} = \mathcal{V}_1 \cup \mathcal{V}_2 \cup \ldots \cup \mathcal{V}_p$*, where* $\mathcal{V}_r = \{\langle v, w \rangle \mid v, w \in V_r\}$*, and the edge set* $\mathcal{E} = \mathcal{E}_1 \cup \mathcal{E}_2 \cup \ldots \cup \mathcal{E}_{p-1}$*, where* $\mathcal{E}_r = \{\langle e, f \rangle \mid e, f \in E_r,\ e = (t, u),\ f = (w, v),\ \langle t, w \rangle \in \mathcal{V}_r,\ \langle u, v \rangle \in \mathcal{V}_{r+1}\}$.

Informally, the vertices of the vertex-exchange graph are all distinct pairs of same-level vertices of the original level graph. Two vertices of a vertex-exchange graph are connected by an edge whenever the corresponding vertices of the original graph are connected by non-adjacent edges. Examples of vertex-exchange graphs are shown in Figure 1.

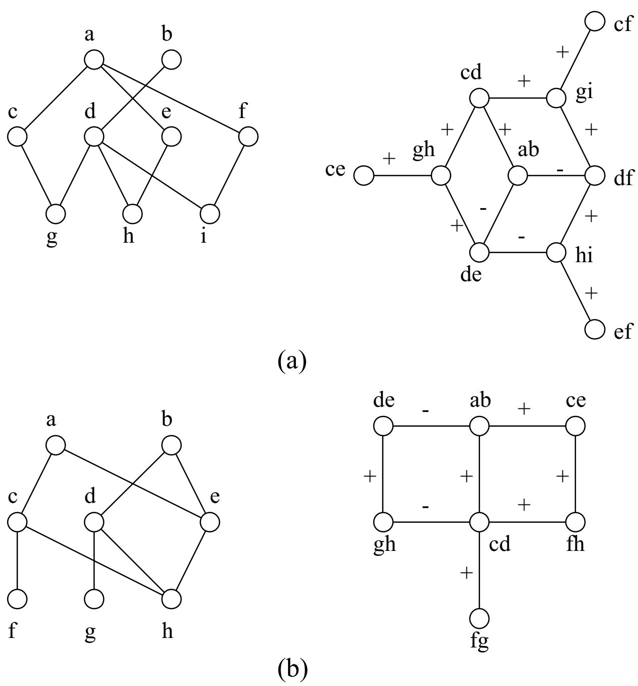


Fig. 1. Examples of vertex-exchange graphs

From the definition of the vertex-exchange graph, we can immediately state some simple properties.

Property 1. If the original graph is a p-level graph then its vertex-exchange graph is also a p-level graph.

From this it follows that the vertex-exchange graph is *bipartite*.

Property 2. The vertex-exchange graph does not contain self-loops. Between two vertices $v = \langle v_1, v_2 \rangle$ and $w = \langle w_1, w_2 \rangle$ there can exist at most two edges. If there exists two edges, then $\{v_1, v_2, w_1, w_2\}$ is the vertex set of a $K_{2,2}$ subgraph. The vertex-exchange graph of a $K_{2,2}$-free level graph is simple.

Given an embedding $\pi : V \rightarrow \mathbb{Z}$ of the original graph, we represent it as a labelling of the vertex-exchange graph as follows. For each vertex $\langle v, w \rangle$ of the vertex-exchange graph, we assign an ordered pair of vertices $ordering(\langle v, w \rangle) = [v, w]$, if $\pi(v) < \pi(w)$, and $ordering(\langle v, w \rangle) = [w, v]$ otherwise. Also, we label each edge $\langle e, f \rangle$ by $label(\langle e, f \rangle) =$ '-', if e and f cross in embedding π, otherwise by $label(\langle e, f \rangle) =$ '+'.

Suppose we have a vertex-exchange graph, and we wish to determine which labellings are valid. It turns out that not all labellings correspond to feasible embeddings of the original level graph. There are the following constraints:

3-cycle — Vertex orderings must correspond to a feasible linear ordering, e.g., for vertices u, v, and w of the original graph, the configuration of vertex orderings $[u, v]$, $[v, w]$, and $[w, u]$ is invalid, because it signifies a "cyclic" linear ordering $\pi(u) < \pi(v) < \pi(w) < \pi(u)$.

vertex-edge — Edge labels and vertex orderings must be consistent. If there are edges (v_1, w_1) and (v_2, w_2) in the original graph then the vertex-exchange graph's edge $(\langle v_1, v_2 \rangle, \langle w_1, w_2 \rangle)$ must have label '+', if the vertex orderings are $[v_1, v_2]$ and $[w_1, w_2]$ or $[v_2, v_1]$ and $[w_2, w_1]$; and by '-' for the 2 other possible vertex orderings.

min-minus — If a level graph is level non-planar then every labelling of its vertex-exchange graph must contain some edges labelled by '-'.

From constraint **min-minus** it follows that if there is a vertex exchange graph with all edges labelled by '+' then the input graph is level planar. We examine now the constraint **min-minus** more closely to determine where unavoidable '-'-edges may occur.

2.2 Odd-Labelled Cycles

First, we describe formally when two vertices of a vertex-exchange graph are *adjacent*. A *path*, a *cycle* and a *connected component* can be defined similarly.

Property 3. Two vertices $v = \langle v_1, v_2 \rangle$ and $w = \langle w_1, w_2 \rangle$ of vertex-exchange graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ are adjacent if vertices v_1 and v_2 are on the same level of G and w_1 and w_2 are on the same level of G and the levels are adjacent and there exist either edges (v_1, w_1) and (v_2, w_2) or (v_1, w_2) and (v_2, w_1) in the input graph.

Examples of even- and odd-labelled cycles can be found in Figure 1. The cycle (ab, cd, gi, df) in Figure 1(a) is odd labelled, but the cycle (ab, cd, gh, de) in Figure 1(b) is even-labelled.

To understand the importance of odd-labelled cycles we will describe here briefly how a level planar embedding can be calculated by an algorithm based on a depth-first search of the labelling of $\mathcal{G}$ of an initial embedding, π, of G. Further details of the level planarity testing and level planar embedding algorithm are given in Sections 3 and 4, respectively.

Firstly, we mark all the vertices of $\mathcal{G}$ as "unvisited." We start from some vertex v of $\mathcal{G}$ and mark it "unchanged" and maintain ordering $[w_1, w_2]$. Next, we pick an adjacent vertex w of v; we keep the ordering $ordering(w) = [w_1, w_2]$ and mark w "unchanged" if (v, w) is a '+'-edge. Otherwise we reverse $[w_1, w_2]$ and mark w "reversed." We repeat the procedure for all the "unvisited" adjacent vertices, u, of w, reversing the vertex ordering of u if (w, u) is labelled '-' and maintaining the ordering if the edge is labelled '+'.

For instance, applying this algorithm for vertex-exchange graph in Figure 1(b), starting from vertex fg, would result in de and gh being marked "reversed." This corresponds exactly to a level planar embedding.

It is not difficult to see that if a vertex-exchange graph contains an odd-labelled cycle then conflicting markings of some vertex of the odd-labelled cycle occur because the two paths along the cycle impose different markings. For example, if we start from vertex ab in Figure 1(a), then gi would get different marking by paths (ab, cd, gi) and (ab, df, gi). The next theorem shows that the odd-labelled cycles in the vertex-exchange graph are the *only* causes of level non-planarity.

Theorem 1. *A level graph G is level planar if and only if the vertex-exchange graph $\mathcal{G}$ of G does not contain any odd-labelled cycles.*

Proof. Level planar $\Rightarrow$ no odd labelled cycles. Every level planar graph has a level planar embedding. The labelling of the vertex-exchange graph $\mathcal{G}$ corresponding to a level planar embedding has all edge labels '+', which means all cycles of $\mathcal{G}$ are trivially even-labelled.

No odd-labelled cycles $\Rightarrow$ level planar. Clearly, if $\mathcal{G}$ does not contain any odd-labelled cycles, it is possible to reorder vertex pairs without any conflicts. Also, due to the way the algorithm works, it is guaranteed that edge and vertex labels match. What needs to be shown, is that the calculated vertex labels do not cause any three-cycle conflicts.

Let us assume that we have a sequence of vertices a, b, and c on some level of the input graph G. To have a three-cycle violation, there must be an even-labelled path $(\langle a, b\rangle, \dots, \langle b, c\rangle)$ and an odd-labelled path (paths) $(\langle a, b\rangle, \dots, \langle a, c\rangle)$ and (or) $(\langle b, c\rangle, \dots, \langle a, c\rangle)$ in the vertex-exchange graph. Next, let x be the vertex of the even-labelled path corresponding to the highest or lowest vertex pair $\langle x_1, x_2\rangle$ of the input graph. Let y and $\langle y_1, y_2\rangle$ be those of an odd-labelled path. Let $(\langle b, c\rangle, \dots, y)$ be an odd-labelled path and $(y, \dots, \langle a, c\rangle)$ be an even-labelled path. From the vertex-exchange graph we construct the input graph which will be level non-planar.

The situation where $(\langle b, c\rangle, \ldots, y)$ is the even-labelled path and $(y, \ldots, \langle a, c\rangle)$ is the odd-labelled path can be proved in a similar fashion and likewise for the cases where $x \equiv y$ or the pairs share only one vertex such as $x_1 \equiv y_2$.

It is worthwhile noting that an alternative proof method of Theorem 1 is to show that the vertex-exchange graphs of each type of minimal level non-planar subgraphs [5] possess at least one odd-labelled cycle. (A minimal level non-planar graph $G = (V, E)$ is a non-planar graph such that $G' = (V, E \setminus \{e\}), \forall e \in E$ is planar.)

In fact, minimal level planar subgraphs are directly related to odd-labelled cycles, as shown by the following theorem.

Theorem 2. *Let $\mathcal{C}$ be an odd-labelled cycle in a vertex-exchange graph and C be the subset of edges of the input graph which are mapped to the edges of $\mathcal{C}$. Then there exists a subset of edges $F \subseteq C$ which is the edge set of a minimal level non-planar subgraph.*

Proof. Consider a graph G which includes exactly one level minimal non-planar subgraph, H. Then, from Theorem 1, the vertex-exchange graph $\mathcal{G}$ of G has at least one odd-labelled cycle. By the removal of any edge from H, the graph G becomes level planar and, therefore, the odd labelled cycle disappears from the vertex-exchange graph $\mathcal{G}$. Hence, every edge of H maps to some edge of the odd-labelled cycle $\mathcal{C}$ of $\mathcal{G}$.

As an illustration of this theorem, consider cycle (cd, gi, df, hi, de, gh) in Figure 1(a). It corresponds to the subgraph with vertex set $\{c, d, e, f, g, h, i\}$ of the original graph. This subgraph is a 2-level minimal non-planar subgraph called a double-claw.

3 Level Planarity Testing by the Vertex-Exchange Graph

The basic idea of the level planarity testing algorithm was given in the previous section. The algorithm consists of calling the depth first search routine (Algorithm 1) for each connected component of a vertex-exchange graph. If each component search returns 'true' then the whole vertex-exchange graph has no odd-labelled cycles and the input graph is level planar.

We have used the following encoding for vertex labels. Each vertex v of the vertex-exchange graph has an attribute $value(v)$ which signifies the ordering of the corresponding vertices of the input graph. Three values are possible for $value(v)$: $unknown$ for not visited, $true$ for the original ordering, and $false$ for reverse ordering. Note that we do not actually change any *edge* labels, because they will finally be all '+' for a level planar graph anyway.

Using the depth first search routine above we can achieve a bound of $O(|V|^2)$ on the running time. Naively, the running time of the algorithm for an arbitrary input graph, is $O(|\mathcal{E}|) = O(|E|^2)$ – since the depth-first search visits each edge exactly once. However, it would be foolish to apply the algorithm to arbitrary

Algorithm 1 *LevelPlanarityDFS*$(\mathcal{G}, v, b)$

```
if value(v) = unknown then
   value(v) = b
   for all vertices w adjacent to v in G do
      if label((v, w)) = '+' then
         result = LevelPlanarityDFS(G, w, b)
      else
         result = LevelPlanarityDFS(G, w, ¬b)
      end if
      if result = false then
         return false
      end if
   end for
else if value(v) ≠ b then
   return false
end if
return true
```

graphs because graphs with $|E| > 2|V| - 4$ cannot be planar by a corollary of Euler's formula for bipartite graphs. A vertex-exchange graph is bipartite and planarity is a necessary condition for level planarity.

Consequently, when applying this algorithm, it makes sense to perform the simple check that $|E| \leq 2|V| - 4$ first.

Asymptotically, this algorithm does not compete with an efficient PQ-tree-based linear-time algorithm by Leipert [8]. However, our algorithm has a good trade-off between efficiency and conceptual simplicity. It can be understood and implemented with reasonably small effort yet it can test level planarity of a 200-vertex graph in no more than 2 seconds[1]. So, we consider our algorithm preferable in situations where the graphs are not very big and a PQ-tree library is not available.

4 Layout Calculation of a Level Planar Graph

In addition to level planarity testing, Algorithm 1 can also be used to calculate the layout of a level planar graph. After the completion of *LevelPlanarityDFS*, all the vertex *value*s are either 'true' or 'false'. The embedding can then be found from these values by using a sorting algorithm.

However, the requirement that the 3-cycle constraints be satisfied is a complication. As we have shown, 3-cycle constraints are automatically satisfied in a connected component of a vertex-exchange graph. The 3-cycle constraints formed by the vertices belonging to different connected components need extra care. Figure 2 provides an example graph where the 3-cycles cause a problem.

[1] All experimental work was carried out on a 300MHz DEC AlphaStation.

For instance, if we assign vertex labels as follows:

$$\begin{aligned} value(\langle a, b\rangle) &= true \Rightarrow value(\langle d, f\rangle) = false \\ value(\langle d, e\rangle) &= true \\ value(\langle e, f\rangle) &= true \end{aligned}$$

then we get a 3-cycle violation since the latter two assignments imply $value(\langle d, f\rangle) = true$.

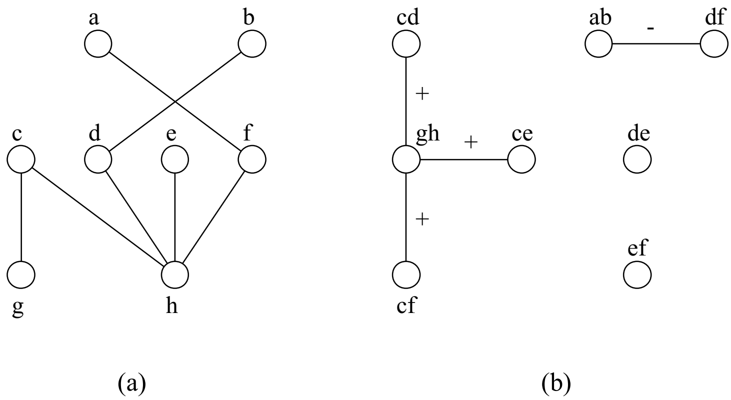


Fig. 2. A graph whose layout calculation has a 3-cycle problem (a); and its vertex-exchange graph (b).

This complication is solved as follows. The vertex-exchange graph is divided into connected components as in the level planarity testing case. Then, a table of all those 3-cycles which possess vertices of different components is constructed. There are 6 fields in the table: identifiers and values of all 3-cycle vertices. For each table record, a mapping from each of the three participating vertices is made. This permits location of an entry of the table of 3-cycles quickly (in almost constant or logarithmic time depending on the implementation of the map).

Another auxiliary data structure is the queue of component assignments. The queue items have fields for vertex identifier, vertex value, and the component the vertex belongs to. The queue is used for identifying the next component which needs processing by *LevelPlanarityDFS*. Initially, the queue is empty.

Now whenever a vertex is assigned a value (line 2 of Algorithm 1), the table is updated accordingly. If the update results in the assignment of a single remaining vertex value constrained by the other two vertex values in a 3-cycle then the remaining vertex identifier with its value and component identifier is inserted into the queue. If a component has been processed by *LevelPlanarityDFS* and the next component has to be selected and there is an item in the queue then the

component corresponding to the queue item is taken as the next one. The first vertex to be assigned and its value is then taken from the queue item instead of taking an arbitrary vertex and value. A minor technical detail is that for each component we have a boolean value indicating whether it is processed or not in order to avoid processing the same component twice.

The enhancements for keeping track of three-cycles unfortunately make the level planar layout algorithm slower than the level planarity testing algorithm. Since the 3-cycle table has $O(|V|^3)$ items then its construction – and, therefore, the whole algorithm – has $O(|V|^3)$ running time.

5 Crossing Minimisation

Crossing minimisation of general level graphs is far more interesting than layout calculation of level planar graphs from a practical perspective. As we observed before, there are three different types of constraints which need to be satisfied when minimising crossings using the vertex-exchange graph. Thus it is natural to suggest Integer Linear Programming (ILP) as a convenient method to satisfy a big variety of constraints. We will see below that properties of the vertex-exchange graph play a role here also.

Jünger *et al.* [6] have proposed an ILP formulation of the crossing minimisation problem. They encode the problem using binary variables for linear ordering. For notational convenience we will refer to a vertex v with $\pi(v) = i$ as vertex i and, similarly, an edge (v, w) with $\pi(v) = i$ and $\pi(w) = j$ as edge (i, j).

$$x_{ij}^r = \begin{cases} 1 & \text{if vertex } i \text{ is placed before vertex } j \text{ on level } r, \\ 0 & \text{if vertex } i \text{ is placed after vertex } j \text{ on level } r. \end{cases}$$

and for the occurrence of a crossing:

$$c_{ijkl}^r = \begin{cases} 1 & \text{if edges } (i,j) \text{ and } (k,l) \text{ cross}, \\ 0 & \text{otherwise}, \end{cases}$$

The inequalities represent 3-cycle constraints and the relation between linear orderings of vertices and crossings (constraints **3-cycle** and **vertex-edge**). The whole ILP formulation is expressed as follows.

Minimise

$$\sum_{r=1}^{p-1} \sum_{(i,j)(k,l) \in E_r} c_{ijkl}^r \tag{1}$$

subject to

$$-c_{ijkl}^r \le x_{jl}^{r+1} - x_{ik}^r \le c_{ijkl}^r \qquad (i,j),\ (k,l) \in E_r,\ j < l \tag{2}$$

$$1 - c_{ijkl}^r \le x_{jl}^{r+1} + x_{ik}^r \le 1 + c_{ijkl}^r \qquad (i,j),\ (k,l) \in E_r,\ j > l \tag{3}$$

$$0 \le x_{ij}^r + x_{jk}^r - x_{ik}^r \le 1 \qquad 1 \le i < j < k \le |V_r| \tag{4}$$

$$x_{ij}^r, y_{ij}^r, c_{ijkl}^r \in \{0,1\}\ . \tag{5}$$

This ILP formulation can find the layout with minimum number of crossings of medium-sized graphs (about 30-40 vertices and edges) in a few seconds. However, the authors suggest the need for additional inequalities in order to develop a practically useful algorithm.

We propose to employ the odd-labelled cycles for additional inequalities. The basic idea for additional constraints is that we switch the labelling of the edges of a vertex-exchange graph deliberately so that all the cycles become even-labelled. For the input graph, this means that we deliberately create some crossings. As the goal is to have the minimum number of crossings in the final layout, we are interested in as few edge-label switchings as possible. A switching of an edge label means that we set a crossing variable $c_{ijkl} = 1$ for the edges (i, j) and (k, l) of the input graph that the edge of the vertex-exchange graph is mapped from.

Fundamental cycles are of use in finding new equalities. A fundamental cycle is defined as follows.

Definition 3. *Given a simple path* $\mathcal{P} = (u, \dots, w)$ *in a spanning tree* T *of* $G = (V, E)$*, then if there is a chord* $(u, w) \in E$*, then* $\mathcal{C} = \mathcal{P} \cup (u, w)$ *is a fundamental cycle of* G*.*

We now make use of the following simple fact regarding fundamental cycles.

Lemma 1. *If all the fundamental cycles of a vertex-exchange graph* $\mathcal{G}$ *are even-labelled then all the cycles of* $\mathcal{G}$ *are even-labelled.*

Proof. Omitted.

For convenience we will change our notation slightly. Let e be an edge belonging to a fundamental cycle $\mathcal{C}$. In terms of the original graph, let $e = (\langle i, k\rangle, \langle j, l\rangle)$ for some i, j, k, and l. We will use c_e to denote c_{ijkl}.

Now the constraints are expressed as follows. For each odd-labelled fundamental cycle $\mathcal{C}$:

$$\sum_{e \in \mathcal{C}} c_e = 2k_{\mathcal{C}} + 1, \qquad 0 \le k_{\mathcal{C}} \le |\mathcal{C}|/2 - 1 \tag{6}$$

and for each even-labelled fundamental cycle $\mathcal{C}$:

$$\sum_{e \in \mathcal{C}} c_e = 2k_{\mathcal{C}}, \qquad 0 \le k_{\mathcal{C}} \le |\mathcal{C}|/2 \tag{7}$$

where $c_e \in \{0, 1\}$ and $k_{\mathcal{C}} \in \mathbb{Z}$.

Equation (6) expresses the constraint that an odd-labelled cycle can have only an odd number of switched labellings; equation (7) constrains to be even the number of switched labellings that an even-labelled cycle can have.

Let $\mathcal{C}$ be an odd labelled cycle. Then it is possible to express constraint (6) without the extra integer variables $k_{\mathcal{C}}$:

$$-|\mathcal{D}| + 1 \le -\sum_{e \in \mathcal{D}} c_e + \sum_{e \in \mathcal{C} \setminus \mathcal{D}} c_e \le |\mathcal{C}| - |\mathcal{D}| - 1$$

$$\forall \mathcal{D} \subset \mathcal{C},\ |\mathcal{D}| = 0, 2, \dots, 2\lfloor |\mathcal{C}|/4 \rfloor \tag{8}$$

In the same way, (7) can be expanded.

$$-|\mathcal{D}|+1 \leq -\sum_{e\in\mathcal{D}} c_e + \sum_{e\in\mathcal{C}\setminus\mathcal{D}} c_e \leq |\mathcal{C}|-|\mathcal{D}|-1$$
$$\forall\mathcal{D}\subset\mathcal{C},\ |\mathcal{D}|=1,3,\dots,2\lfloor(|\mathcal{C}|-2)/4\rfloor+1 \quad (9)$$

In general, this transformation does not seem to be very efficient because the number of inequalities for one cycle becomes $\sum_{i=1}^{|\mathcal{C}|/4}\binom{|\mathcal{C}|}{2i} = O(2^{|\mathcal{C}|})$. Nonetheless, practical results show that for each fundamental odd-labelled cycle $\mathcal{C}$ the single inequality $\sum_{e\in\mathcal{C}} c_e \geq 1$ alone decreases the number of branch-and-bound nodes visited by the ILP solver.

5.1 Crossing Number Bounds

We have implemented the new ILP formulation, and we have observed that the size of the branch-and-bound tree with new equalities and inequalities is approximately half of that of the original formulation. One of the reasons why the new constraints have such influence can be explained by the improved *lower bound* calculation. It is easy to see that the LP-relaxation of the original ILP (1)–(5) results always with objective value 0, all $c_{ijkl}=0$, and all $x_{ij}=0.5$. The new cycle constraints do not allow this, since they force some crossing variables to take non-zero values.

Thus, repeated addition of constraints derived from the vertex-exchange graph provide a sequence of improved lower bounds for the crossing number.

We have also observed in our work that the number of crossings in any embedding of G is always less than or equal to the number of odd-labelled fundamental cycles. Thus we make the following conjecture.

Conjecture 1. The crossing number of a graph, G, is bounded from above by the number of odd-labelled fundamental cycles of its vertex-exchange graph $\mathcal{G}$.

6 Conclusions

In this paper we have proposed a new technique for analysing proper level graphs. The technique – which we call the vertex-exchange graph – admits in an obvious way a method for testing the planarity of (proper) level graphs. Although not asymptotically optimal its conceptual straightforwardness and ease of coding may make the algorithm a viable alternative to known optimal algorithms. Building on this algorithm we have shown how the layout of a planar level graph can be determined. To the best of our knowledge this is the first algorithm that does not lay out such a graph in a level-by-level fashion.

We presented a further application of this idea by using it in an ILP formulation of crossing minimisation. This yielded an improving sequence of lower bounds on the crossing number. Although we do not show it here, under certain conditions these constraints can be shown to be facet defining for the polytope.

Though we have shown several uses of the vertex-exchange graph, we believe others applications exist. For instance, it would be interesting to investigate how the vertex-exchange graph can be employed by heuristic crossing minimisation algorithms. Also, our conjecture that the number of odd-labelled fundamental cycles is an upper bound on the crossing number of the levelled graph would be an interesting and important result.

References

1. G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. *Graph Drawing, Algorithms for the Visualization of Graphs*. Prentice Hall, 1999.
2. P. Eades and D. Kelly. Heuristics for drawing 2-layered networks. *Ars Combinatoria*, 21-A:89–98, 1986.
3. P. Eades and K. Sugiyama. How to draw a directed graph. *Journal of Information Processing*, 13(4):424–437, 1990.
4. P. Eades and N. C. Wormald. Edge crossings in drawings of bipartite graphs. *Algorithmica*, 11:379–403, 1994.
5. P. Healy and A. Kuusik. Characterisation of level non-planar graphs by minimal patterns. Technical Report UL-CSIS-98-4, University of Limerick, 1998.
6. M. Jünger, E. K. Lee, P. Mutzel, and T. Odenthal. A polyhedral approach to multi-layer crossing minimization problem. In *Graph Drawing, 5th International Symposium, GD '97, Rome, Italy, September 1997*, volume 1353 of *Lecture Notes in Computer Science*, pages 13–24. Springer-Verlag, 1997.
7. M. Jünger and P. Mutzel. Exact and heuristic algorithms for 2-layer crossing minimization. In *Graph Drawing, Symposium on Graph Drawing, GD '95. Passau, Germany, September 20–22, 1995*, volume 1027 of *Lecture Notes in Computer Science*, pages 337–348. Springer-Verlag, 1995.
8. S. Leipert. *Level planarity testing and embedding in linear time.* PhD thesis, Institut für Informatik, Universität zu Köln, 1998.
9. K. Sugiyama, S. Tagawa, and M. Toda. Methods for visual understanding of hierarchical system structures. *IEEE Transactions on Systems, Man, and Cybernetics*, 11(2):109–125, 1981.
10. J. Utech, J. Branke, H. Schmeck, and P. Eades. An evolutionary algorithm for drawing directed graphs. In *Proceedings of the 1998 International Conference on Imaging Science, Systems, and Technology (CISST'98)*, pages 154 – 160, 1998.

Using Sifting for k-Layer Straightline Crossing Minimization

Christian Matuszewski, Robby Schönfeld, and Paul Molitor

Institute for Computer Science, University Halle-Wittenberg,
Kurt-Mothes-Strasse 1, D-06120 Halle(Saale), Germany
{matuszew,schoenfe,molitor}@informatik.uni-halle.de

Abstract. We present a new algorithm for k-layer straightline crossing minimization which is based on sifting that is a heuristic for dynamic reordering of decision diagrams used during logic synthesis and formal verification of logic circuits. The experiments prove sifting to be very efficient. In particular it outperforms the traditional layer by layer sweep based heuristics known from literature by far when applied to k-layered graphs with $k \geq 3$.

1 Introduction

Directed graphs are commonly used to represent information in many fields such as software engineering, project management, and social sciences. A good visualization of this information is necessary to get general ideas of relations.

A common method for drawing directed graphs was introduced by [STT81]. This approach consists of four phases. In the first step, the directed graph is made acyclic by temporarily reversing some edges. Let $G = (V, E)$ be the resulting graph. Then, set V of the nodes is partitioned into k layers $V_1, \ldots, V_k$ $(k \geq 1)$ such that for any edge $(u, v) \in E$ with $u \in V_i$ and $v \in V_j$, the inequation $i < j$ holds. Subset V_i is called ith layer. $G = (V_1 \cup \ldots \cup V_k, E)$ is called a k-layered directed graph. In the third step, the vertices within each layer are permuted targeting minimization of the number of crossings. In the last step, the vertices and edges are placed according to the permutations computed in step 3. The vertices of each layer are placed on a horizontally line, the layer i is placed above the layer $i+1$ for all $i = 1, \ldots, k-1$, and the edges are drawn as straight lines.

In this paper, we focus on step 3. We assume that graph G has been made proper in step 2, i.e., that for any edge $(u, v) \in E$ with $u \in V_i$ and $v \in V_j$ the equation $j - i = 1$ holds. This can be achieved by inserting dummy vertices on edges connecting vertices on non-neighboring layers. The problem we have to solve in step 3 is to find a permutation π_i for every layer V_i such that the number of edge crossings is minimized. Note that the number of edge crossings only depends on the permutations of the vertices and not on the exact positions of the vertices because edges are drawn as straight lines. Unfortunately, the problem of finding a permutation π_i for every layer V_i such that the number of crossings is minimal is NP-hard even for 2-layered directed graphs [GJ83] which

J. Kratochvíl (Ed.): GD'99, LNCS 1731, pp. 217–224, 1999.

we call *two sided crossing minimization problem* in the following. Furthermore, the problem of 2-layer straightline crossing minimization remains NP-hard when the permutation of one layer is fixed [EW94a]. We call this problem the *one sided crossing minimization problem*.

In Section 2 we shortly review heuristics known from the literature. Section 3 presents the new approach which is a standard technique for minimization of binary decision diagrams (BDDs) during logic synthesis and formal verification of logic circuits. The experimental procedure of this section only concentrates on the one sided crossing minimization problem. We generalize sifting to general k-layer straightline crossing minimization in Section 4.

2 Algorithms Known from the Literature

Most methods to reduce the crossings in a k-layered directed graph use a technique called the *layer-by-layer sweep* which works as follows. First, an initial vertex ordering for every layer is computed. Then, for $i = 2, \ldots, k$, the vertex ordering of layer V_{i-1} is held fixed while the vertices of layer V_i are reordered to reduce crossings. After that, the method sweeps back holding fixed the permutation of layer V_i and permute the vertices in layer V_{i-1}. This process is repeated until no further refinement can be achieved. With this approach the k-layer crossing minimization problem is reduced to a series of one sided crossing minimization problems.

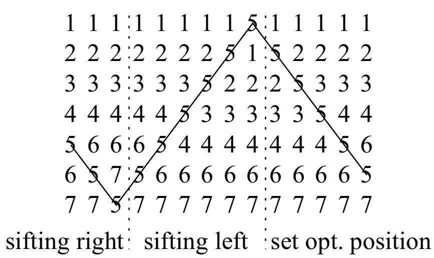
Let us consider the one sided crossing minimization problem in more detail. Without loss of generality, let the permutation π_1 of layer V_1 be fixed. For every pair of vertices $u, v \in V_2$ we define the *crossing number* c_{uv} as the number of edge crossings that edges incident to u make with edges incident to v if $\pi_2(u) < \pi_2(v)$ holds. Thus, $\text{cross}(G, \pi_1, \pi_2) = \sum_{\pi_2(u)<\pi_2(v)} c_{uv}$ is the number of crossings in the straightline drawing of the 2-layered graph $G = (V_1 \cup V_2, E)$ with respect to the permutations π_1 and π_2. An obvious lower bound of $\text{cross}(G, \pi_1, \pi_2)$ is given by $L = \sum_{\pi_2(u)<\pi_2(v)} \min\{c_{uv}, c_{vu}\}$. Experiments in [JM97] have proven this lower bound to be very tight to the optimum.

The most popular heuristics for one sided crossing minimization are the barycenter [STT81] and the median heuristic [EW94b]. The other heuristics known from literature, e.g., split [EK86], greedy-insert [EK86], and greedy-switch heuristic [EK86], are mostly outperformed by barycenter and median heuristic as shown by [JM97]. It is worth remarking that Jünger and Mutzel presented a very efficient branch and cut algorithm for one sided crossing minimization practically applicable to graphs up to 60 vertices in their paper.

3 Sifting for One Sided Crossing Minimization

Sifting was first introduced by Rudell [Rud93] to reduce the number of vertices in reduced ordered binary decision diagrams (ROBDDs). This algorithm can be easily adapted to the one sided crossing minimization problem. Assume that

permutation π_1 of layer V_1 is fixed. We choose a vertex v from layer V_2 to put it on a position which minimizes the number of crossings. The ordering of all other vertices in layer V_2 remains fixed. In order to do so, vertex v is moved to the rightmost position by repeatedly swapping it with its right neighbor. After reaching the rightmost position, v is moved to the left. When v reaches the leftmost position, it is set to the (locally) optimal location. For illustration, please see Figure 1 on the left.




```
failcount ← 0
repeat
    foreach v_i ∈ φ do sifting(v_i)
    if fail then
        reverse(φ)
        failcount ← failcount + 1
    foreach v_i ∈ φ do sifting(v_i)
    if fail then
        failcount ← failcount + 1
    reverse(φ)
until failcount ≥ MAXFAILS
```

Fig. 1. `Sifting` of vertex 5 (left) and `Global Sifting` (right).

To find the best position for a given vertex, we have to compute the number of crossings after each swapping. Assume we have to swap the two vertices u and v with u being the left neighbor of v. Then, after the swapping step there are $\text{cross}_{after\_swapping} = \text{cross}_{before\_swapping} - c_{uv} + c_{vu}$ crossings. With that, sifting of one vertex takes $\mathcal{O}(n)$ resulting in an overall runtime of $\mathcal{O}(n^2)$, where n is the number of vertices.

We consider three different methods for choosing the next vertex to be sifted. Method 1 uses a preassigned vertex order of the layer, e.g., from left to right. A more accurate approach seems to be to sift the vertices according to their degrees, i.e., sift the vertices with high indegree, first (method 2). The third method investigated by us randomly chooses the vertices.

Experimental Results for one Sided Crossing Minimization

Our computational experiments include the barycenter heuristic, the median heuristic, the greedy-insert heuristic, the greedy-switch heuristic, the split heuristic, and the sifting heuristics (method 1 up to method 3). For our experiments we used the algorithms implemented in the AGD library [MGB+98] which is based on LEDA [MN99]. We have used the program `random_bigraph` of the Stanford GraphBase [Knu93] with the same parameters as chosen by Jünger and

Mutzel [JM97]. We could reproduce the results of Jünger and Mutzel exactly except for the median heuristic and the barycenter heuristic. The differences with the median heuristic are due to the fact that the implementation in AGD uses the *average median* from [Mäk90] rather than the original median heuristic from [EW94b]. Our results are always better than those given in [JM97]. The slight deviation from the results with the barycenter heuristic are possibly due to minor differences in sorting methods.

In the first experimental run for one sided crossing minimization, we have considered sparse graphs with an increasing number of vertices, 10 samples for each type of graphs. Such instances are among the most interesting in practical applications. We have used sparse graphs with $|E| = |V_1| + |V_2|$, i.e., on the average, two edges are incident to each vertex. Figure 2 gives the relative size of the average number of crossings taken over all sampled instances of the given graph type in percentage of the minimum number of crossings, which has been computed by the branch and cut algorithm for one sided crossing minimization of Jünger and Mutzel [JM97]. The curves prove that all three sifting methods are close to the optimum for sparse graphs and that sifting dominates the heuristics known from literature for that class of graphs. The running time of the sifting algorithm is below 1 second [1] for 100+100-graphs, i.e., graphs $G = (V_1 \cup V_2, E)$ with $V_1 = V_2 = 100$.

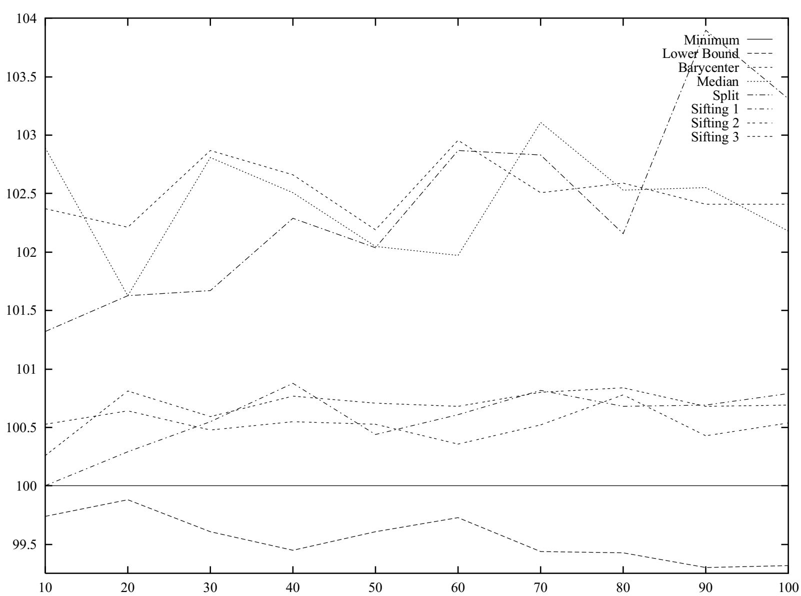


Fig. 2. Results for one sided crossing minimization, sparse graphs, increasing number of vertices, 10 samples.

[1] 296 MHz SUNW, UltraSPARC-II, 512 MB.

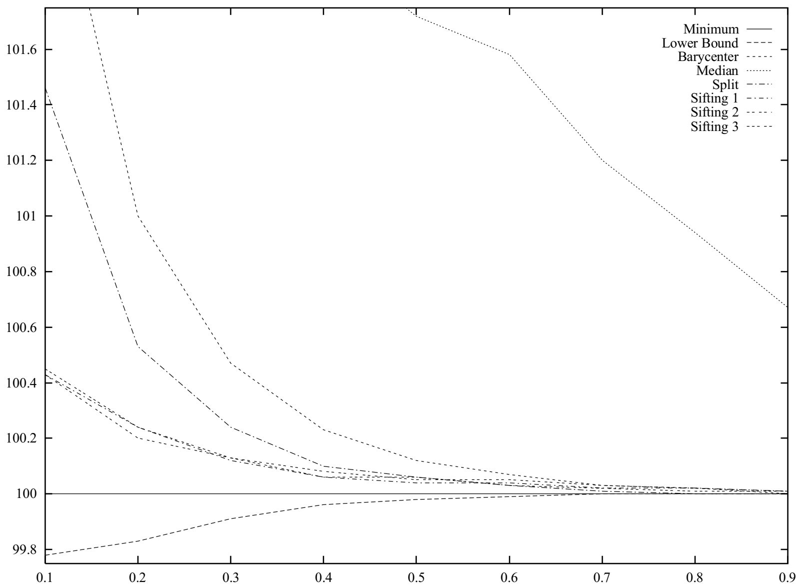


Fig. 3. Results for one sided crossing minimization, 20+20-graphs, increasing edge density, 100 samples.

The next experimental run we have made considers 20+20-graphs with increasing edge densities from 10 up to 90 percent. Figure 3 gives the result for 100 samples for each type of graph. It shows that for one sided crossing minimization sifting dominates the other heuristics for small densities. In the case of high densities, the barycenter heuristic and the split heuristic are as efficient as sifting is.

To sum it up it can be said that sifting applied to one sided crossing minimization is a very efficient heuristic and the vertex order used during sifting does not influence the quality of the heuristic very much.

4 Extending Sifting to k-Layered Directed Graphs

As stated in [JM97], there is no real need for heuristics for one sided crossing minimization up to 60 vertices in the permutable layer as the optimum solution can be computed fast by the branch and cut algorithm. So, we should concentrate on the general problem, namely on the k-layer straightline crossing minimization problem for $k \geq 2$.

Sifting can obviously be extended to k-layer straightline crossing minimization by using layer by layer based sifting (see Section 2). Another method which we call `Global Sifting` is to keep a list $\phi = (v_1, \ldots, v_n)$ of the vertices of V in descending order of the degree of the vertices and to sift each vertex in its associated layer according to this order. Vertices with high degree are handled, first.

If no improvement is made, we reverse the vertex order of list ϕ and memorize it as a fail , otherwise we use the same sequence for the next trial again. After these two loops passed, the vertex order is inverted in any case and the cycle is repeated if the number of fails is less than a defined maximum count. Despite the possibility of better results, we have not taken care about this constant and set it to zero in all experimental runs. Note that in this algorithm optimization is not only realized layer by layer. Figure 1 on the right presents the algorithm in detail.

Experimental Results for k-Layered Graphs

In our experiments, we have considered directed graphs up to 12 layers with different numbers of vertices and edge densities. All heuristics, except global sifting, are iterated between the k layers until a local optimum is obtained.

For small sparse 2-layered graphs $G = (V_1 \cup V_2, E)$ with $|E| = |V_1| + |V_2|$ sifting is slightly better than the barycenter method, whereas barycenter dominates sifting for larger sparse 2-layered graphs. The insufficient quality of sifting for increasing number of vertices seemed to be due to the fact that the density decreases when the number of vertices increases. A sparse 10+10-graph, e.g., contains 20 edges resulting in a density of 20 percent, whereas a 100+100-graph which contains 200 edges leads to a density of only 2 percent. We repeated our experimental run for increasing number of vertices with a constant edge density of 20 percent. The results gave evidence of our assumption. Sifting and global sifting are slightly better than the barycenter method in this case. However, this slight improvement does not justify the higher computation time of sifting.

For more than 2 layers, global sifting dominates all the other heuristics by far. This fact is illustrated by Figure 4 and 5. Here, the horizontal curve at the 100 percentage level represents the results for global sifting. In the experiments, we used k*b-graphs, i.e., k-layered directed graphs $G_k = (V_1 \cup \ldots \cup V_k, E)$ with $|V_i| = b$ for all $1 \leq i \leq k$.

In a first experimental run, we varied the number of vertices in sparse graphs with 4 layers. For this, we extended the concept of sparse graphs from 2 layers to k layers by only considering k*b-graphs $G = (V_1 \cup \ldots \cup V_k, E)$ with $|E| = 2 \cdot (k-1) \cdot b$. Then, on the average, a vertex must have 2 neighbors in each adjacent layer. Figure 4 shows the results for sparse k*b-graphs with $k = 4$ and $10 \leq b \leq 100$ taken over 10 samples. Surprisingly, layer by layer based sifting is much worse than the barycenter, median, and split heuristic. However, global sifting is very efficient and outperforms all the other methods.

The next experimental run, we have made, considers k*b-graphs with higher densities. Although all the heuristics converge to the true optima when the edge density is increased, global sifting dominates the other heuristics. In particular it is much more efficient for dense graphs than all the other heuristics, too. Detailed information on this experiment are included in the technical report appeared at Martin-Luther-University.

Finally, we studied the influence of the number of layers on the quality of the crossing minimization (see Figure 5). For this, we have used sparse graphs with 10 vertices per layer. The number of layers is rising from 2 up to 12 layers. It

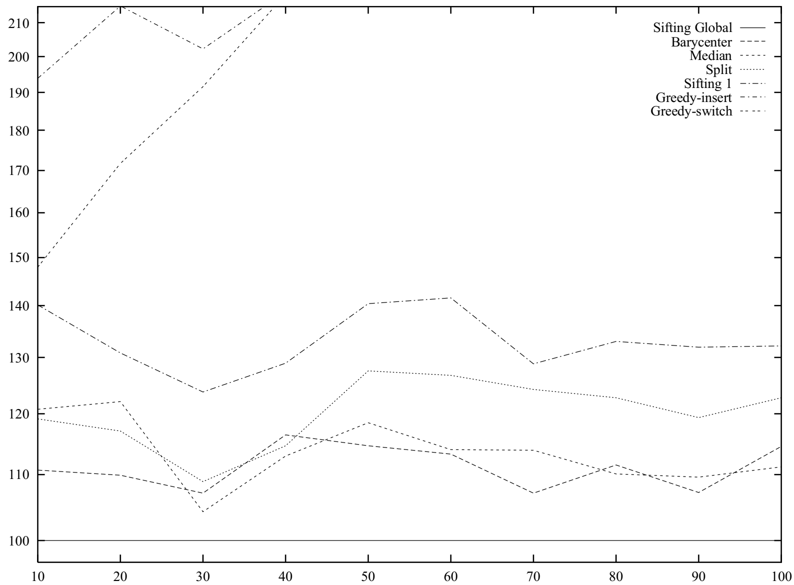


Fig. 4. Results for 4-layer crossing minimization, sparse graphs, increasing number of vertices, 10 samples.

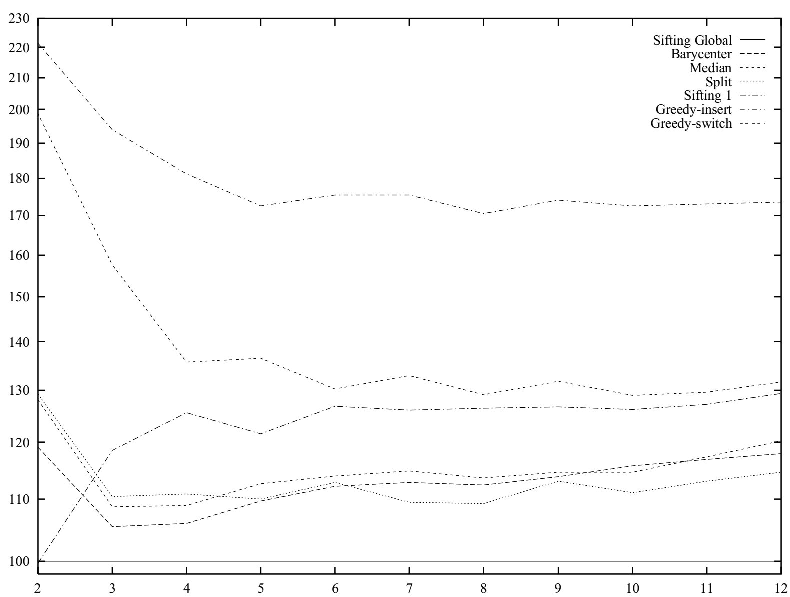


Fig. 5. Results for k-layer crossing minimization, sparse graphs, 10 vertices per layer, increasing number of layers, 100 samples.

can be observed that the layer by layer sweep based heuristics lose performance against global sifting for increasing number of layers.

5 Conclusions

We have presented a new approach for k-layer straightline crossing minimization which has been proven to be efficient. The experiments lead to the conclusion that global sifting dominates the heuristics known from literature for one sided and k-layered crossing straightline minimization with $k \geq 3$.

Acknowledgments

We would like to thank Petra Mutzel, Thomas Ziegler, and Stefan Näher for helpful discussions concerning the AGD library and LEDA.

References

EK86. P. Eades and D. Kelly. Heuristics for reducing crossings in 2-layered networks. *Ars Combin.*, 21.A:89–98, 1986.

EW94a. P. Eades and S. Whitesides. Drawing graphs in two layers. *Theoretical Computer Science*, 131:361–374, 1994.

EW94b. P. Eades and N. C. Wormald. Edge crossings in drawings of bipartite graphs. *Algorithmica*, 11(4):379–403, 1994.

GJ83. M. R. Garey and D. S. Johnson. Crossing number is NP-complete. *SIAM Journal on Algebraic and Discrete Methods*, 4:312–316, 1983.

JM97. M. Jünger and P. Mutzel. 2-Layer straightline crossing minimization: Performance of exact and heuristic algorithms. *J. Graph Algorithms Appl.*, 1(1):1–25, 1997.

Knu93. D. E. Knuth. *The Stanford GraphBase: A Platform for Combinatorial Computing.* Addison-Wesley, Reading, MA, 1993.

Mäk90. E. Mäkinen. Experiments on drawing 2-level hierarchical graphs. *Internat. J. Comput. Math.*, 36:175–181, 1990.

MGB+98. P. Mutzel, C. Gutwenger, R. Brockenauer, S. Fialko, G. W. Klau, M. Krüger, T. Ziegler, S. Näher, D. Alberts, D. Ambras, G. Koch, M. Jünger, C. Buchheim, and S. Leipert. A library of algorithms for graph drawing. In S. H. Whitesides, editor, *Proceedings of the 6th International Symposium on Graph Drawing (GD '98)*, volume 1547 of *Lecture Notes in Computer Science*, pages 456–457. Springer, 1998. Project home page at `http://www.mpi-sb.mpg.de/AGD/`.

MN99. K. Mehlhorn and S. Näher. *The* LEDA *Platform of Combinatorial and Geometric Computing.* Cambridge University Press, 1999. Project home page at `http://www.mpi-sb.mpg.de/LEDA/`.

Rud93. R. Rudell. Dynamic variable ordering for ordered binary decision diagrams. In *Proc. International Conf. on Computer-Aided Design (ICCAD)*, pages 42–47, November 1993.

STT81. K. Sugiyama, S. Tagawa, and M. Toda. Methods for visual understanding of hierarchical system structures. *IEEE Transactions on Systems, Man and Cybernetics*, 11(2):109–125, February 1981.

On 3-Layer Crossings and Pseudo Arrangements

Farhad Shahrokhi[1]* and Imrich Vrťo[2]**

[1] Department of Computer Science, University of North Texas
P.O.Box 13886, Denton, TX, USA
farhad@csci.unt.edu
[2] Department of Informatics, Institute of Mathematics
Slovak Academy of Sciences
P.O.Box 56, 840 00 Bratislava, Slovak Republic
vrto@savba.sk

Abstract. Let $G = (V_0, V_1, V_2, E)$ be a 3-layer graph. The 3-layer drawings of G in which V_0, V_1, and V_2 are placed on 3 parallel lines and each edge in E is drawn using one straight line segment, are studied. A generalization of the linear arrangement problem which we call the 3-layer pseudo linear arrangement problem is introduced, and it is shown to be closely related to the 3-layer crossing number. In particular, we show that the 3-layer crossing number of G plus the sum of the square of degrees asymptotically has the same order of magnitude as the optimal solution to the 3-layer linear arrangement problem. Consequently, when G satisfies certain (reasonable) assumptions, we derive the first polynomial time approximation algorithm to compute the 3-layer crossing number within a multiplicative factor of $O(\log n)$ from the optimal.

1 Introduction

The planar crossing number problem is the problem of placing the vertices of a graph in the plane and drawing the edges with curves, to minimize the number of edge crossings [20]. This problem is known to be NP-hard [9] and has been extensively studied in graph theory [23,16], and theory of VLSI [13]. One of the most important aesthetic objectives in drawing graphs is to have a small number of crossings [17], and therefore the crossing minimization problems have been frequently studied by the graph drawing community e.g. [4,5,11,15].

Let $G = (V, E)$ be an undirected graph with the vertex set V and the edge set E. G is called a k-layer graph, if a partition of V into k sets $V_0, V_1, ..., V_{k-1}$ exists so that any edge in E has one end point in V_i and the other end point in V_{i+1} for some $i = 0, 1, 2, ..., k-2$. Thus, any bipartite graph is a $2-$layer graph. If $G = (V, E)$ is a $k-$layer graph, then we write $G = (V_0, V_1, ..., V_{k-1}, E)$, where $\{V_0, V_1, ..., V_{k-1}\}$ is the partition of V into k disjoint sets. Let $G =$

* This research was supported by NSF grant CCR-9528228. A part of this work was completed when the first author was visiting DIMACS in the Fall of 1998.
** This research was supported by grant No. 95/5305/277 of The Slovak Grant Agency.

J. Kratochvíl (Ed.): GD'99, LNCS 1731, pp. 225–231, 1999.

$(V_0, V_2, ..., V_{k-1}, E)$ be a k-layer graph, $k \geq 2$, and let $L_0, L_1, ..., L_{k-1}$ be k parallel lines in the plane. A *$k-$layer drawing* [1,4,12,15,21] of G consists of placing the vertices of V_i into distinct points on L_i, $i = 0, 1, 2..., k-1$, and then drawing each edge using a straight line segment connecting the points representing the end vertices of the edge. The objective is to minimize the number of crossings between the edge pairs. Note that any $k-$layer drawing of G is identified by a one to one function

$$D : V_0 \cup V_1 ... \cup V_{k-1} \rightarrow \Re,$$

where $\Re$ is the set of non-negative real numbers. In particular, note that the restriction of D to $V_i, 0 \leq i \leq k-1$, specifies the order in which the vertices of V_i will appear on L_i. When $k = 2$ the corresponding drawing is called a *bipartite drawing* and the problem of minimizing the number of crossings is called the *bipartite crossing number problem* [5,11,15,21,19].

Computing the bipartite crossing number is NP-hard [9][1] and despite a great deal of research which was done on this problem, no polynomial time approximation algorithm had been known for this problem. Very recently a polynomial time approximation algorithm with performance guarantee of $O(\log n)$ from the optimal was discovered for approximating the bipartite crossing number of a large class of graphs on n vertices [19]. Nonetheless, no efficient approximation algorithm for minimizing the number of crossings in a k-layer drawing has been known for $k \geq 3$. The main results in [19] were obtained by relating the bipartite crossing number problem to the linear arrangement problem which is another well known problem in theory of VLSI. In this paper we develop a general framework to study the 3-layer crossing number problem by relating it to a very general version of the linear arrangement problem which we call the *pseudo-arrangement problem.* In particular, we derive tight upper and lower bounds for the 3-layer crossing number in terms of the arrangement values. The ratio of the main term of the upper bound to the main term in the lower bound is only 3, and the error term is the sum of the square of degrees in G. The result is interesting, since it indicates that the number of edge crossings is closely related to the *length* of the drawing which is defined by the pseudo-arrangement problem. Consequently, we derive the first polynomial time approximation algorithm for the 3-layer crossing problem with the performance guarantee of $O(\log n)$ from the optimal, provided that the graph satisfies certain conditions.

2 Basic Concepts and Notations

Let $G = (V, E)$ be a graph, we denote by d_v the degree of $v \in V$. Throughout this paper $G = (A, B, E)$ denotes a bipartite graph on the partite sets A and B, and the edge set E. $G = (V_0, V_1, V_2, E)$ denotes a 3-layer graph, with vertex set $V = V_0 \cup V_1 \cup V_2$, $|V| = n$, and the edge set E.

[1] Technically speaking, the NP-hardness of the problem was proved for multigraphs, but it is widely assumed that it is also NP-hard for simple graphs.

For $G = (V_0, V_1, V_2, E)$, let G_0 and G_2 denote, respectively, the induced subgraphs of G on the vertex sets $V_0 \cup V_1$, and $V_2 \cup V_1$. Note that G_0 and G_2 are bipartite graphs. Note that there is no edge in G with one end vertex in V_0 and the other in V_2. For any $v \in V_1$, let $d_{v,0}$ and $d_{v,2}$ denote the degrees of v, in G_0 and G_2, respectively, and note that $d_{v,0} + d_{v,2} = d_v$.

A 3-layer drawing of $G = (V_0, V_1, V_2)$ is a one-to-one function

$$D : V_0 \cup V_1 \cup V_2 \to \Re.$$

Let D_i denote the restriction of D to V_i, $0 \leq i \leq 2$. Note that D_i, $0 \leq i \leq 2$ specifies the order in which the vertices in V_i are placed on the line L_i. Observe that (D_0, D_1) is a bipartite drawing for G_0 and (D_1, D_2) is a bipartite drawing for G_2. For any $e \in E$, let $cr_0(e)$ denote the number of crossings of e with other edges in the drawing (D_0, D_1), and $cr_2(e)$ denote the number of crossings of e with other edges in the drawing (D_1, D_2) . We define cr_0 and cr_2 to be the total number of crossings in the drawings (D_0, D_1) and (D_1, D_2), respectively, and define cr_D to be the total number of edge crossings in D. Thus, $cr_D = cr_0 + cr_2$. The *3-layer crossing number* of G is the minimum number of crossings of edges over all 3-layer drawings of G.

A *linear arrangement* (LA) of a graph $G = (V, E)$ is a one to one function $f : V \to \{1, 2, ...|V|\}$. The linear arrangement problem is to find a LA so that $\sum_{uv \in E} |f(u) - f(v)|$ is minimized [2,3,7,10,18]. This problem is known to be NP-hard but can be approximated in polynomial time using a variety of algorithms[7,10,18]. Crucial to our work are generalizations of this problem defined for bipartite and 3-layer graphs.

For $x, y \in \Re$, let (x, y) denote the open interval between x and y. Let $G = (A, B, E)$. A pseudo linear arrangement (PLA) for G is a one to one function $f : A \cup B \to \Re$ so that $f(B) = \{1, 2..., |B|\}$. Hence, any vertex in B is assigned a unique integer which is at most equal to $|B|$. Let $ab \in E$, with $f(a) < f(b)$. We define the *length* of e, denoted by L_f^e, to be

$$\sum_{x \in B, f(x) \in (f(a), f(b))} d_x.$$

We define the length of f, denoted by L_f, to be $\sum_{ab \in E} L_f^e$. The pseudo linear arrangement problem is to find a PLA of minimum length. We denote this minimum value by $\bar{L}_G$. It follows from the recent work on spreading matrices [7,18] that for any graph on n vertices $\bar{L}_G$ can be approximated to within a factor of $O(\log n)$ from the optimal in polynomial time.

A *3-layer pseudo linear arrangement* (3PLA) of $G = (V_0, V_1, V_2, E)$ is a one to one function $f : V_0 \cup V_1 \cup V_2 \to \Re$, so that $f(V_1) = \{1, 2, 3, ..., |V_1|\}$. Note that we may view any 3PLA f of G as a 3-layer drawing of G. Let f_i denote the restriction of f to V_i, $0 \leq i \leq 2$, and note that that (f_0, f_1) and (f_1, f_2) are PLAs of G_0 and G_2, respectively. Let $ab \in E$, we define the *length* of e, denoted by L_f^e to be $L_{(f_0, f_1)}^e$, provided that e is an edge in G_0, otherwise, we define L_f^e to be $L_{(f_1, f_2)}^e$. Note that for any edge e in G_0

$$L_f^e = \sum_{x \in V_1, f(x) \in (f(a), f(b))} d_{x,0}$$

whereas, for any e in G_2,

$$L_f^e = \sum_{x \in V_1, f(x) \in (f(a), f(b))} d_{x,2}.$$

The length of f, denoted by L_f, is defined to be $\sum_{e \in E} L_f^e$. The 3-layer pseudo linear arrangement problem is to find a 3PLA for G which has the minimum length. We denote this minimum value by L_G. Let f be a 3PLA of G. For $v \in V_0 \cup V_2$, let $u_1, u_2, ..., u_{d_v}$ be its neighbors in the set V_1 satisfying $f(u_1) < f(u_2) < ... < f(u_{d_v})$. We define the *median vertex* of v, denoted by $med(v)$, to be $u_{\lceil \frac{d_v}{2} \rceil}$.

3 Arrangements and 3-Layer Drawings

Let $G = (V_0, V_1, V_2, E)$ and D be a 3-layer drawing of G. We assume throughout this paper that the vertices of V_0 are placed on the line $y = 0$, vertices of V_1 are placed on the line, $y = 1$, and vertices of V_3 are placed on the line $y = 2$. Moreover, since the number of crossings only depends on the order of vertices, we will assume throughout this paper that the vertices of V_1 are placed into the points

$$(1, 1), (2, 1), ..., (|V_1|, 1).$$

Note that for any $v \in V$, $D(v)$ is the x-coordinate of v, and that D is a 3PLA of G. In particular, note that for any $v \in V_0 \cup V_2$, $med(v)$ is the vertex $u_{\lceil \frac{d_v}{2} \rceil}$, where $u_1, u_2, ..., u_{d_v}$ are neighbors of v in the set V_1 satisfying $D(u_1) < D(u_2) < ... < D(u_{d_v})$.

Theorem 1. *Let D be a 3-layer drawing of $G = (V_0, V_1, V_2, E)$, then*

$$cr_D \geq \frac{1}{2}\Big(L_G - \sum_{v \in V_0 \cup V_2} \Big\lfloor \frac{d_v}{2} \Big\rfloor d_v \Big).$$

Proof. To show the lower bound on cr_D, consider the bipartite drawing (D_0, D_1) of G_0. Let $v \in V_0$ with $d_v \geq 2$, and let $u_1, u_2, ..., u_{d_v}$ be its neighbors with $D(u_1) < D(u_2) < ... < D(u_{d_v})$. Let i be an integer, $1 \leq i \leq \lfloor d_v/2 \rfloor$, and let u be a vertex in V_1 so that that $D(u_i) < D(u) < D(u_{d_v - i+1})$. Observe that u generates $d_{u,0}$ crossings on the edges $u_i v$ and $u_{d_v-i+1} v$, if it is not adjacent to v; similarly, u generates $d_{u,0} - 1$ crossings on the edges $u_i v$ and $u_{d_v-i+1} v$, if it is adjacent to v. Thus,

$$cr_0(u_i v) + cr_0(u_{d_v-i+1} v) \geq \sum_{D(u) \in (D(u_i), D(u_{d_v-i+1}))} d_{u,0} - d_v.$$

Hence, for $v \in V_0$ with $d_v \geq 2$,

$$\sum_{i=1}^{d_v} cr_{D_0}(u_i v) \geq \sum_{i=1}^{\lfloor \frac{d_v}{2} \rfloor} \sum_{D(u) \in (D(u_i), D(u_{d_v - i+1}))} d_{u,0} - \left\lfloor \frac{d_v}{2} \right\rfloor d_v.$$

We conclude by taking the sum over all $v \in V_0, d_v \geq 2$ that,

$$2cr_0 \geq \sum_{v \in V_0} \sum_{i=1}^{d_v} cr_0(u_i v) \geq \sum_{v \in V_0} \sum_{i=1}^{\lfloor \frac{d_v}{2} \rfloor} \sum_{D(u) \in (D(u_i), D(u_{d_v - i+1}))} d_{u,0} - \sum_{v \in V_0} \left\lfloor \frac{d_v}{2} \right\rfloor d_v.$$

Using a similar approach we obtain

$$2cr_2 \geq \sum_{v \in V_2} \sum_{i=1}^{d_v} cr_2(u_i v) \geq \sum_{v \in V_2} \sum_{i=1}^{\lfloor \frac{d_v}{2} \rfloor} \sum_{D(u) \in (D(u_i), D(u_{d_v - i+1}))} d_{2,0} - \sum_{v \in V_2} \left\lfloor \frac{d_v}{2} \right\rfloor d_v.$$

Define a 3PLA by: $f(v) = D(v)$ for all $v \in V_1$, and $f(v) = med(v) + \epsilon_v$ for all $v \in V_0 \cup V_2$, where ϵ_v is an infinitely small value. Let $e = vx$, where $v \in V_0 \cup V_2$, and $x = med(v)$, then $L_f^e = 0$. It follows that

$$L_f = \sum_{v \in V_0 \cup V_2} \sum_{i=1}^{\lfloor \frac{d_v}{2} \rfloor} \sum_{D(u) \in (D(u_i), D(u_{d_v - i+1}))} d_{u,0}.$$

Hence, $2cr_D \geq L_f - \sum_{v \in V_0 \cup V_2} \lfloor \frac{d_v}{2} \rfloor d_v$, and the claim follows by observing that $L_G \leq L_f$. □

Theorem 2. *Let f be a 3PLA of $G = (V_0, V_1, V_2, E)$, then there is 3-layer drawing D of G so that*

$$cr_D \leq \frac{3}{2} L_f + \sum_{v \in V_1} d_v^2.$$

Proof. Define a new 3PLA by: $D(v) = f(med(v))$ for any $v \in V_0 \cup V_2$, and $D(v) = f(v)$, for any $v \in V_1$. If two vertices are placed at the same location, we separate them by placing an arbitrary small distance between them.

$$L_D \leq L_f.$$

To prove the upper bound on cr_D, we estimate from above the number of crossings on any edge incident to a vertex $v \in V_0 \cup V_2$. The sum of the number of crossings can be shown to be at most $3/2 L_D + \sum_{v \in V_1} d_v^2$ using the method to derive the lower bound. □

By Theorems 1 and 2, in order to construct a 3-layer drawing with small number of crossings, one only needs to find a 3PLA of G with small value. Unfortunately, it is not known how to compute exactly or even approximate

the 3-layer arrangement problem, in polynomial time. Nonetheless, we can show that if G_0 and G_2 do not have vertices of degree zero, then, this problem "essentially" becomes the pseudo linear arrangement problem, provided that G is degree bounded. Let $G = (V_0, V_1, V_2, E)$, we denote the maximum degree among all vertices in V_1 by Δ_1.

Lemma 1. *Let $G = (V_0, V_1, V_2, E)$, and let $\bar{f}$ be any PLA for the bipartite graph $G = (V_0 \cup V_2, V_1, E)$. Then, $\bar{f}$ induces a 3PLA, denoted by f, so that*

$$L_f \leq L_{\bar{f}}.$$

Moreover, if $d_{v,0} > 0$ and $d_{v,2} > 0$, for any $v \in V_1$, then

$$\frac{L_{\bar{f}}}{\Delta_1} \leq L_f.$$

We can now present our main result.

Theorem 3. *Let $G = (V_0, V_1, V_2, E)$ so that $|E| > (2 + \epsilon)|V|$, for a positive ϵ. Assume that $d_{v,0} > 0$ and $d_{v,2} > 0$, for all $v \in V_1$, and that Δ_1 is bounded by a constant. Then, the 3-layer crossing number can be approximated to within a factor of $O(\log n)$ from the optimal in polynomial time.*

Proof. The conditions on E and Δ_1 can be used to show that for any 3-layer drawing D,

$$cr_D = \Omega\Big(\sum_{v \in V_0 \cup V_2} d_v^2\Big).$$

Hence, by Theorem 1 $cr_D = \Omega(L_G)$. It remains to construct an $O(\log n)$ times optimal 3PLA, denoted by f, since then by Theorem 2 we can construct the desired drawing D. To construct f, we first construct a $O(\log n)$ times optimal PLA using the algorithm in [18]. By Lemma 1 this gives a $O(\log n)$ times optimal 3PLA, or f, since Δ_1 is bounded by a constant. □

References

1. Catarci, T.: The assignment heuristics for crossing reduction. IEEE Transactions on Systems, Man and Cybernetics **25** (1995) 515–521
2. Chung, F. R. K.: On optimal linear arrangements of trees. Computers and Mathematics with Applications **10** (1984) 43–60
3. Díaz, J.: Graph layout problems. In International Symposium on Mathematical Foundations of Computer Sciences. Lecture Notes in Computer Science, Vol. 629. Springer-Verlag, Berlin Heidelberg New York (1992) 14–21
4. Di Battista, J., Eades, P., Tamassia, R., Tollis, I. G.: Algorithms for drawing graphs: an annotated bibliography. Computational Geometry **4** (1994) 235–282
5. Eades, P., Wormald, N.: Edge crossings in drawings of bipartite graphs. Algorithmica **11** (1994) 379–403
6. Eades, P., Whitesides, S.: Drawing graphs in 2 layers. Theoretical Computer Science **131** (1994) 361–374

7. Even, G., Naor, J. S., Rao, S., Schieber, B.: Divide-and-Conquer approximation algorithms via spreading matrices. In 36th Annual IEEE Symposium on Foundation of Computer Science. IEEE Computer Society Press (1995) 62–71
8. Even, G., Naor, J. S., Rao, S., Schieber, B.: Fast approximate graph partition algorithms. In 8th Annual ACM-SIAM Symposium on Discrete Algorithms. ACM Press (1997) 639–648
9. Garey, M. R., Johnson, D. S.: Crossing number is NP-complete. SIAM J. Algebraic and Discrete Methods **4** (1983) 312–316
10. Hansen, M.: Approximate algorithms for geometric embeddings in the plane with applications to parallel processing problems. In 30th Annual IEEE Symposium on Foundation of Computer Science. IEEE Computer Society Press (1989) 604–609
11. Jünger, M., Mutzel, P.: Exact and heuristic algorithm for 2-layer straight line crossing number. In 3rd Symposium on Graph Drawing'95. Lecture Notes in Computer Science, Vol. 1027. Springer-Verlag, BerlinHeidelberg New York (1996) 337–348
12. Jünger, M., Lee, E. K., Mutzel, P., Odenthal T.: A polyhedral approach to the multi-layer crossing number problem. In 5th Symposium on Graph Drawing'97. Lecture Notes in Computer Science, Vol. 1353. Springer-Verlag, Berlin Heidelberg New York (1997) 13-24
13. Leighton, F. T.: Complexity issues in VLSI. MIT Press, Massachusetts (1983)
14. May, M., Szkatula, K.: On the bipartite crossing number. Control and Cybernetics **17** (1988) 85–98
15. Mutzel, P.: An alternative method to crossing minimization on hierarchical graphs. In 4th Symposium on Graph Drawing'96. Lecture Notes in Computer Science, Vol. 1190. Springer-Verlag, Berlin Heidelberg New York (1997) 318-333
16. Pach, J., Agarwal, K.: Combinatorial Geometry. John Wiley & Sons Inc., New York (1995)
17. Purchase, H.: Which aesthetics has the greatest effect on human understanding? In 5th Symposium on Graph Drawing'97. Lecture Notes in Computer Science, Vol. 1353. Springer-Verlag, Berlin Heidelberg New York (1997) 248-261
18. Rao, S., Richa, A.:, New approximation techniques for some ordering problems. In: 9th Annual ACM-SIAM Symposium on Discrete Algorithms. ACM Press (1998) 211-225
19. Shahrokhi, F., Sýkora, O., Székely, L. A., Vrťo , I.: On bipartite crossings, largest biplanar subgraphs, and the linear arrangement problem. In Workshop on Algorithms and Data Structures'97. Lecture Notes in Computer Science, Vol. 1272. Springer-Verlag, Berlin Heidelberg New York (1997) 55–68
Extended version will appear in *SIAM Journal on Computing* as: On bipartite drawings and the linear arrangement problem.
20. Shahrokhi, F., Sýkora, O., Székely, L.A., and Vrťo , I.: Crossing number problems: bounds and applications. In: Bárány, I., and Böröczky, K. (eds): Intuitive Geometry. Bolyai Society Mathematical Studies, Vol 6. Akadémia Kiadó, Budapest (1997) 179-206
21. Sugiyama, K., Tagawa, S., Toda, M.: Methods for visual understanding of hierarchical systems structures. IEEE Transactions on Systems, Man and Cybernetics **11** (1981) 109–125
22. Warfield, J.: Crossing theory and hierarchy mapping. IEEE Transactions on Systems, Man and Cybernetics **7** (1977) 502–523
23. White, A. T., and Beineke, L. W.: Topological graph theory. In: L. W. Beineke, L. W., and R. J. Wilson, R. J. (eds.): Selected Topics in Graph Theory. Academic Press, New York (1978) 15–50

Visualizing Algorithms for the Design and Analysis of Survivable Networks

Ala Eddine Barouni[1], Ali Jaoua[2], and Nejib Zaguia[3]

[1] University of Tunis, department of computer science, Tunisia
ala.barouni@fst.rnu.tn
[2] King Fahd University of Petroleum and Minerals, department of computer science, Dhahran, Arabie Saoudia
ajaoua@ccse.kfupm.edu.sa
[3] School of Information Technology and Engineering, Ottawa, Ontario, Canada
zaguia@site.uottawa.ca

Abstract. We present algorithms for the drawing of survivable telecommunication networks. The visualization of telecommunication networks is a very important problem. For some specific rings in a network, we may have a high traffic. The network designers may decide to add more equipment to the nodes (sites) of these rings in order to increase the performance of the network. Therefore, one of the most important properties of the survivable telecommunication network, is that rings should be easily recognizable. Given a ring cover of survivable telecommunication networks, we provide three techniques for drawing a ring cover. We should mention that all these drawings should respect many criteria in order to preserve the readability of the drawing. As in most of the graph drawing algorithms, the area used for the drawing is very important, our proposed algorithms produce drawings that require $O(n^2)$ area, where n is the number of nodes in the ring cover. These drawings are clear and easy to understand by the network designers.

1 Introduction

The problem of drawing a graph in the plane has received increasing attention recently due to the large number of applications stated in [1]. For example automatic layout of pert diagrams [2], layout algorithm for data flow diagrams [3] and layout algorithm of entity-relationships diagrams [4]. The design and analysis of telecommunication network is a very important area (for more details see [5], [6] and [7]). In this paper, we study techniques for visualizing telecommunication networks. We are motivated to design a network which (1) satisfies the traffic requirements, (2) can survives failures, and (c) the cost of the network is minimum. A network is *survivable* if it can survive the failure of a link *e*, that is, the removal of the link *e* does not disconnect the network, and the traffic that was originally supported by the link *e*, can be accommodated by another path. The *"Multi-Ring Architecture"* is considered

J. Kratochvíl (Ed.): GD'99, LNCS 1731, pp. 232-241, 1999.

to be a cost-effective survivable network because of its simplicity and improved survivability (see [8] and [9]).

Consider a network N represented by a graph $G = (V, E)$, where V is the set of nodes representing the sites and E is a set of links representing the electrical links between nodes. A *ring R* in G is defined to be a cycle consisting of nodes $n_1, n_2, ..., n_p \in V$. For some specific rings in a network, we may have a high traffic. The network designers may decide to add more equipment to the nodes (sites) of these rings in order to increase the performance of the network. Therefore, one of the most important properties of Multi-Ring Architecture, is that rings should be easily recognizable. We focus on drawing networks on the computer screen so that properties of the Multi-Ring Architecture can easily be viewed. Similar to graph drawing, many criteria should be considered in order to enhance the readability. Usually, a general optimization method are used, i.e., minimize the crossings, the area, the number of bends (in orthogonal drawings), and the number of slopes (in polyline drawings). Readability is also our concern, because we have noticed, so far, that unnecessary crossings may create rings that do not exist in the original network. In this paper, we will address the issue of visualizing survivable telecommunication networks by presenting multi-ring architecture drawing techniques.

We define a *Ring Cover C* as a set of rings that covers all links in a network [10]. For a network N represented by a graph G, given a ring cover C, a *contact node c* is a node of G which belongs to the intersection of at least two rings of C. The *resolution* is a constant defined by the user which is considered to be the minimal distance between any two nodes in the drawing of the ring cover. The resolution rule is that the nodes must be kept far enough from each other, so that, the human eye can tell them apart. The resolution rule prevents the drawing algorithm from arbitrary scaling down the picture. The problem is defined as follows: let N be a telecommunication network represented by a graph $G = (V, E)$, where V is the set of nodes representing the sites of switches and E is a set of links representing the electrical wires or optical fiber links between nodes. Corresponding to the graph G, a $|V| \times |V|$ matrix T is defined so that: the entries $T_{i,j}$ denote the amount of traffic between the node i and the node j, where $1 \leq i, j \leq |V|$. In [10], some techniques are provided to find a ring cover in survivable networks by using the matrix T as input. Thus, for a network N, we assume that the ring cover C is given and therefore our task is to deal with the ring cover drawing. We shall present three different techniques of ring cover drawing. They take into account many criteria in order to produce layouts that are clear and easy to understand. Since the nodes of the network correspond to sites, they have geographic coordinates. Hence, the network can be drawn naturally with little effort. However, for some complicated structure of networks, the important properties that designers are interested in, such as rings, are not displayed.

In [11] and [12], three techniques of ring cover drawing are provided:

1. Inside Drawing : Each ring is drawn inside other rings.
2. Outside Drawing : Each ring is drawn outside of other rings.
3. Mixed Drawing : Each ring can be drawn outside or inside other rings.

These three techniques are based on the following assumptions:

1. Any two adjacent rings share only one contact node.
2. The *ring-contact node graph* [11] is a tree and therefore it contains no cycles.

They proposed two open problems:

1. Drawing a ring cover when any two adjacent rings share one contact node and the ring-contact node graph is not a tree.
2. Drawing a ring cover when rings share more than one contact node.

In [13] and [14], we presented two techniques (Inside and outside) of ring covers drawing in order to deal with the open problems presented in [11].

In this paper we focus on expanding the conditions of the ring cover drawing. Our aim is to present a new technique to draw the ring cover, called mixed drawing. Before presenting the mixed drawing algorithm, we should present the inside and the outside drawing, since the mixed drawing uses both the inside and the outside drawing. Our research includes the case treated in [11], [13] and [14], augmented by the following contributions:

1. The ring cover contains rings that share more than one contact node.
2. The ring-contact node graph, which is considered to be the underlying structure of the ring cover is not a tree.
3. A combination of both cases: the ring cover contains rings which share more than one contact node and the ring-contact node graph has cycles.
4. The design and the implementation of a new system for drawing telecommunication networks based on the Multi-Ring Architecture. The system invites the user to enter the ring cover through its underlying structure and then generates the picture of the network.

2 Inside Drawing

In this section, we will describe the inside drawing technique which consists of drawing each ring inside the other one. We will present our approach which consists of extending the drawing of the ring cover to other types of networks. We treat the case where any two adjacent rings in the ring cover can share more than one contact node, and we consider the possibility of having *ring-cycles* in the ring cover. Our approach consists of extending the drawing of the ring cover to other types of networks in which we consider the case where the rings could share more than one contact node and the ring cover contains *ring-cycles*. Before going into details, let us define some concepts:

A path P which connect two contact nodes c_i and c_j is called a *free* path if and only if c_i and c_j are the only contact nodes in this path.

A *ring-cycle* $RC = \{R_1, cn_1, R_2, cn_2, ..., R_i, cn_i, R_{i+1}, cn_{i+1}\}$, where $R_1 = R_{i+1}$ and $cn_1 = cn_{i+1}$, is a set of rings R's and special contact nodes cn's, so that rings form a cycle in the ring cover, that is:

- Any two adjacent rings R_j and R_{j+1} share exactly one contact node cn_j, $j = 1, ..., i$.
- And in each ring R_j, $j = 2, ..., i+1$, there exists at least one free path between two distinct special contact nodes cn_{j-1} and cn_j belonging to the ring R_j.

Let a ring $R = \{n_1, n_2, ..., n_p\}$ be an ordered sequence of its nodes, we call two nodes n_i and n_k $(i \neq k)$ adjacent in R, if there exists a direct link e between n_1 and n_2 within the

ring R. Let $R_X = \{n_1, n_2, ..., n_p\}$ and $R_Y = \{m_1, m_2, ..., m_q\}$ be two rings, such that R_X and R_Y are pairwise compatible [11], and we suppose that they share k contact nodes $\{c_1, c_2, ..., c_k\}$. A multi-contact node mc is the set $\{c_1, c_2, ..., c_k\}$ of contact nodes, such that c_i, c_{i+1} are adjacent nodes in R_X and in R_Y, where $1 < i < k-1$.

Let $MC = \{mc_1, ..., mc_n\}$ be the set of all multi-contact nodes in the ring cover C and $mc_i \in MC$ be the multi-contact node between two rings R_X and R_Y. We say R_X and R_Y are *strictly compatible* if $mc_i \cap mc_j = \emptyset$ for $j = 1, 2, ..., n$ and $j \neq$ i.

Our approach consists of specifying a new underlying structure of the ring cover which will be considered later on as the basis of our drawing. We assume first, that any two adjacent rings in the ring cover are strictly compatible, and we shall introduce the *generalized-ring-contact node graph* considered to be the new underlying structure of the ring cover. We define the *generalized-ring-contact node graph* $G_1(V_1, E_1)$ as follows:

V_1 contains five types of vertices :

1. Rings in the ring cover, denoted by R_i.
2. Special contact nodes that belong to the ring-cycle, denoted by cn_i.
3. Contact nodes that do not belong to ring-cycle, denoted by c_i.
4. Ring-cycles in the ring cover, denoted by RC_i.
5. Mutli-contact nodes, denoted by mc_i.

E_1 contains five types of edges defined as follow:

1. $\{(R, c) : R \in C,$ and $c \in R\}$.
2. $\{(RC, cn) : cn \in RC\}$.
3. $\{(cn, R) : R \in C, cn \in R, cn \in RC$ and $R \notin RC\}$.
4. $\{(RC, R) : R \in C$ and $R \in RC\}$.
5. $\{(R, mc) : R \in C$, and nodes of mc are in $R\}$.

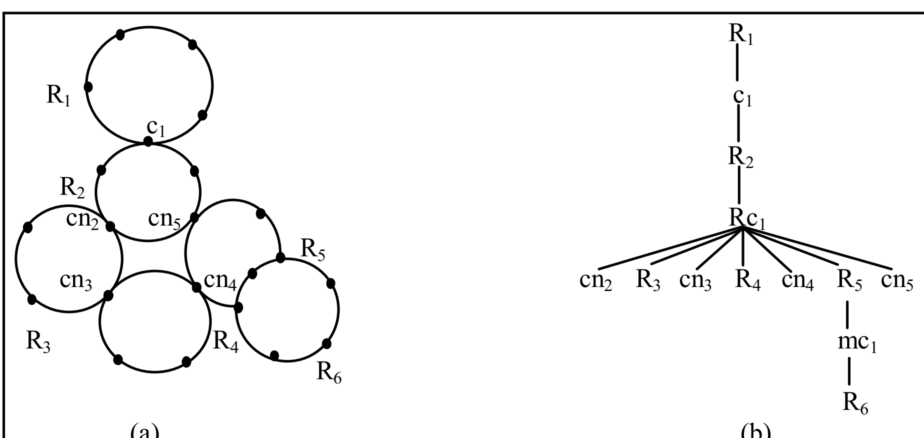


Fig. 1. (a) Ring cover; (b) *Generalized*-ring-contact node graph

In our drawing we focus only on the case where the generalized-ring-contact node graph G_1 is a tree (see figure 1(b)), we call it T_1. Our algorithm traverses the tree T_1 two times : The first traversal of the tree T_1 is in postorder fashion. It consists of computing the radius of the circles corresponding to the rings and the ring-cycles in the ring cover. The second traversal of the tree T_1 is in preorder fashion. It consists of placing the parent node and its children in the right location.

Using the above procedure, we conclude the following theorem:
Theorem 1:
Given a ring cover C and its generalized-ring-contact node tree, the inside drawing algorithm *InsideDraw* [15] produces a ring cover drawing such that :

1. there are no crossings;
2. the area required by the drawing is $O(n^2)$, where n is the number of nodes in C;
3. the time complexity is $O(n + m^2)$, where n is the number of nodes in the ring cover and m is the number of the rings in the ring cover.

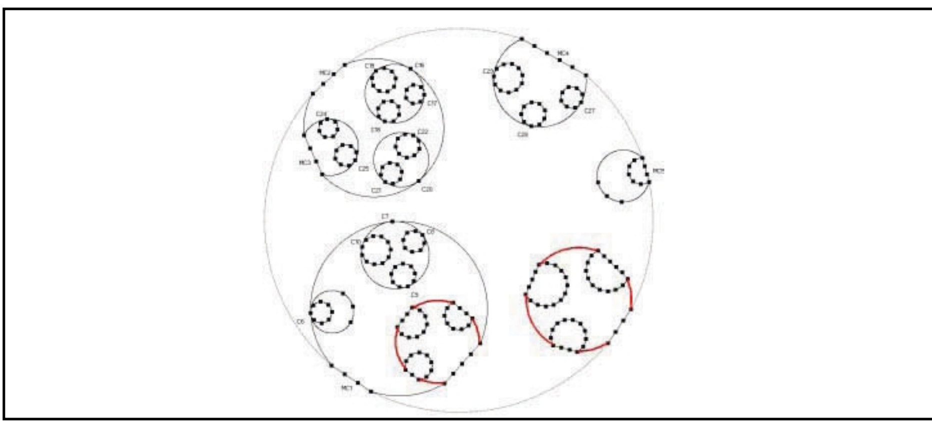


Fig. 2. The Ring cover contains ring-cycles and rings that share more than one contact node with other rings.

3 Outside Drawing

We present in this section the outside drawing technique which consists of drawing each ring outside the other rings. Given a ring cover C and its generalized-ring-contact node tree T_1 (presented in section 2), our algorithm draws rings outside each other, such that the resolution rule is respected, no crossings are generated, and a picture of the ring cover is produced in an optimal area. Our algorithm traverses the generalized-ring-contact node tree twice: The first traversal of the tree T_1 is in postorder fashion. It consists of computing the radius of the circles corresponding to the rings and the ring-cycles in the ring cover. The second traversal of the tree T_1 is in preorder fashion. It consists of placing the parent node and its children in the right position.

Using the above procedure, we conclude the following theorem:

Theorem 2:
Given a ring cover C and its generalized-ring-contact node tree, the outside drawing algorithm *OutsideDraw* [15] produces a ring cover drawing such that:

1. there are no crossings;
2. the area required by the drawing is $O(n)^2$, where n is the number of nodes in the ring cover.
3. the time complexity is $O(n + m^2)$, where n is the number of nodes in the ring cover and m is the number of the rings in the ring cover.

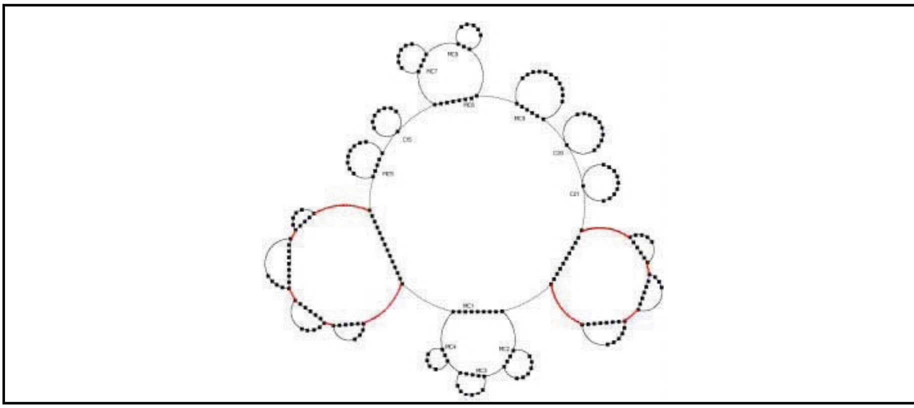


Fig. 3: Ring cover contains rings which share more than one contact node and contains two ring-cycles.

4 Mixed Drawing

In the first part of this section, we will present the mixed drawing technique which uses both inside and outside drawing algorithm for the ring cover drawing. We introduce the mixed drawing for the following reason. First, because it is impossible to draw rings outside each other, when more than two rings share the same multi-contact node. The second reason, is that, the mixed drawing algorithm uses less area than the outside drawing algorithm.

4.1 Basic idea

In the inside drawing, we are able to draw rings when more than two rings share a multi-contact node. Our idea consists of drawing some subtrees of T_1 by the inside algorithm, and other subtrees by the outside algorithm. First, we traverse the tree T_1 in preorder fashion, if we encounter a ring node u, such that it contains a child which is a multi-contact node or a ring-cycle, then the ring corresponding to u and its subtree will be drawn by the inside drawing algorithm, else if u does not have any child

which is either a multi-contact node or a ring-cycle, then *u* and its children will be drawn one outside the other.

4.2 Algorithm

The algorithm *MixDraw_LabelNode* will traverse the tree T_1 in preorder fashion. If we encounter a node *u* which has at least one child which is either a multi-contact node or a ring-cycle, we assign a value true to *INSIDE* associated to the node *u* and all the nodes in the subtree of *u*. This means that, all the rings and the ring-cycles of the subtree of *u* will be drawn inside each other. If all the children of *u* are different than a ring-cycle and a multi-contact node, we assign the value false to *INSIDE* associated to the node *u*

Algorithm *MixedDrawLabelNode(v)*;
Input: Ring cover *C*, and its generalized-ring-contact node tree T_1 with the root *v*.
Output: An update of the label *INSIDE* associated to the rings and the ring-cycles.
Begin
1. If ((*v* is a ring that contains at least one child which is a multi-contact node **or** ring-cycle**) or** *v* is a ring-cycle**)) then**

- Assign the value true to *INSIDE* associated to the node *v* and to all the nodes in the subtree of *v*; **return;**

EndIf;
2. If (*v* is a ring**) and (**all the children of *v* are contact nodes**) then**
Begin

- Assign the value false to *INSIDE* associated to the node *v*;
- **For** all the children u_i of *v* **do** *MixedDraw_LabelNode* (u_i);

EndIf;
3. If (*v* is a contact node **) then**

- **For** all the children u_i of *v* **do** *MixedDraw_LabelNode* (u_i);

EndIf;
End.

The time complexity of *MixedDraw_LabelNode* algorithm is of order $O(n)$, where *n* is the number of nodes in the ring cover. We process (by means of updating the *INSIDE* label) each node in T_1 only one time.
In the second traversal of T_1, the algorithm starts with the root *v* of T_1. If *v* is a ring or a ring-cycle which *INSIDE* value is true, then we call the *InsideDraw* algorithm to compute the radius of the circles corresponding to the rings and the ring- cycles in T_1. If the value of *INSIDE* is equal to false, then the node *v* and all its children will be drawn each one outside the other. By doing so we mix both drawings: the inside drawing and the outside drawing to obtain a new drawing called the mixed drawing.

Algorithm *MixedDraw(v)*;
Input: Ring cover C, and its generalized-ring-contact node tree T_1 with the root v.
Output : Radius of the circles corresponding to the rings and the ring-cycles in C.
Begin
1. If ((v is a ring **or** v is a ring-cycle**) and (***INSIDE(v)* is true**))**
then
- *InsideDraw(v)*; **Return**;

EndIf;
2. If ((v is a ring **) and (***INSIDE(v)* is false**)) then**
Begin
- **For** all the children u_i of v **do** *MixedDraw(u_i)*;
- *OutsideDraw(v) { we place v and its children one outside the other }*

EndIf;
3. If (v is a node representing a contact node **) then**
- **For** all the children u_i of v **do** *MixedDraw(u_i)*;

EndIf;
End.

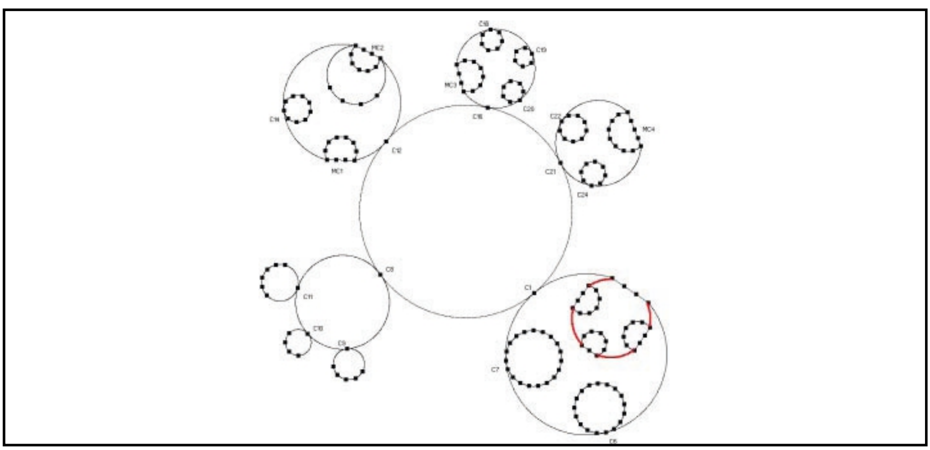


Fig. 4. A picture of a ring cover generated by the system. We notice that if a ring u has at least one child which shares more than one contact node with other rings or a ring-cycle, then u and all the nodes of its subtree will be drawn one inside the other.

Using the above procedure, we conclude the following theorem:
Theorem 3
Given a ring cover C and its generalized-ring-contact node tree T_1, the algorithm *MixedDraw* produces a ring cover drawing such that:

1. There are no crossings;
2. the area required by the drawing is $O(n^2)$, where n is the number of nodes in C;
3. the time complexity is $O(n + m^2)$, where n is the number of nodes in the ring cover and m is the number of the rings in the ring cover.

5 Conclusion and Open Problems

In this paper, we presented three techniques for drawing a ring cover of survivable telecommunication networks: the inside drawing, the outside drawing and the mixed drawing. Our contribution to this subject was to expand the drawing of the ring cover to more complicated structures of survivable telecommunication networks. In fact, we presented an improved version for the outside and the inside drawing in order to deal with the case where the rings share more than one contact node. We proposed the ring-multi-contact node graph as a new underlying structure for the ring cover. Then we have considered the case where the ring cover contains ring-cycles in which we have presented the extended-ring-contact node graph as the new underlying structure for the ring cover. Finally we considered the case where the ring cover contains ring-cycles and rings that share more than one contact node. We also introduced the generalized-ring-contact node graph as the final underlying structure for the ring cover. While dealing with the outside drawing technique, we encountered the problem that occurs when at least three rings share the same multi-contact node, that is why we proposed a new approach for the mixed drawing algorithm.

In order to guarantee the readability of the drawing, we have to respect the following criteria:

1. No crossing is allowed, because if it is the case, we may visualize rings that do not exist in the original network.
2. Rings should be easily recognizable in the picture.
3. The resolution rule should be respected (for visibility purposes).
4. Use of minimal area.

Our proposed drawing algorithms produce drawings that require $O(n^2)$ area, where n is the number of nodes in the ring cover. Besides our theoretical research, we have done an experimental work which consist of implementing the previously designed algorithm. For efficiency purposes, we used the object-oriented language called “Borland Delphi”.

There are still some issues which have not been addressed yet and that we consider them as being open problems, two of them are described here:

1. Are there other reasonable ways to draw ring covers ?
2. How can we draw ring cover when the generalized-ring-contact node graph is not a tree ?

6 Acknowledgement

We would like to thank Professor Ioannis G. Tollis for his help and the interesting discussion we had about this subject.

References

1. G. Di Battista, P. Eades, R. Tamassia and I.G. Tollis. "Algorithms for Automatic Graph Drawing: An Annotated Bibliography," Department of computer Science, Brown University, Technical Report, 1993.
2. G. Di Battista, E. Pietrosanti, R. Tamassia, and I.G. Tollis, "Automatic Layout of PERT Diagrams with XPERT, " Proc. IEEE Workshop on Visual Languages (VL'89).
3. C. Batini, E. Nardelli, and R. Tamassia, "A Layout Algorithm for Data Flow Diagrams". IEEE Trans Software Eng., vol. 12 1986.
4. C. Batini, M. Talamo, and R. Tamassia, "Computer Aided Layout of Entity-Relationships Diagrams" The Journal of Systems and Software, vol. 4, 1984.
5. J.C.Shah , "Restoration Network Planning Tool", Proc. 8th Annual Fiber Optic Engineers Conf. April 21, 1992.
6. W.D.Grover, B.D.Venables, J.H. Sandham, and A.F. Milne, "Performance Studies of a Self-Healing Network Protocol in Telecom Canada Long Haul Networks " Proceedings of IEEE GLOBECOM 1990.
7. H.Sakauchi, Y.Nishimura, and S.Hasegawa, " A Self-Healing Network with an economical Spare-Channel Assignment " IEEE GLOBECOM 1990.
8. Tsong-Ho Wu and R. C. Lau , "A Class of Self-Healing Ring Architectures for SONET Network Application" IEEE Trans. on Communication, Vol. 40, No. 11 Nov. 1992.
9. T.H. Wu, D.J. Collar, and R.H. Cardwell, "Survivable Network Architectures for Broad-band Fiber Optic Networks: Model and Performance Comparison" IEEE journal of Lightwave. Technology, Vol. 6, No. 11, November 1988.
10. L.M. Gardner, M. Heydari, J. Shah, I.H. Sudborough, I.G. Tollis, and C. Xia, "Techniques for Finding Ring Covers in Survivable Networks" Proceedings of IEEE GLOBECOM 1994.
11. Ioannis G. Tollis and Chunliang Xia. "Drawing Telecommunication Networks". DIMACS International Workshop, Graph Drawing'94. Princeton, New Jersey, USA, October 1994. Proceedings.
12. Ioannis G. Tollis and Chunliang Xia. "Graph drawing algorithms for the design and analysis of telecommunication networks". Graph Drawing'93, Proceedings of the ALCOM International workshop on graph drawing Sèvres France.
13. Ala Eddine Barouni, Ali Jaoua and Nejib Zaguia. "Drawing Algorithms for telecommunication networks". In 3rd CGC workshop on Computational Geometry. Brown University, Providence, RI, USA. October 1998.
14. Ala Eddine Barouni, Ali Jaoua and Nejib Zaguia. "Planar Drawing Algorithms of survivable telecommunication networks". In Proceedings of the Japan Conference on Discrete and Computational Geometry. Tokai University, Japan. December 1998.
15. Ala Eddine. Barouni "Drawing Algorithm for survivable telecommunication networks", Master Theses at the university of Ottawa Canada. April 1997.

LayoutShow: A Signed Applet/Application for Graph Drawing and Experimentation

System Demonstration

Lila Behzadi

Department of Computer Science
York University
4700 Keele Street, North York
Ontario M3J 1P3, Canada
lila@cs.yorku.ca

Abstract. LayoutShow is a Java-based multi-threaded applet/application for experimentation with graph drawing algorithms, particularly, force-directed algorithms. The motivation behind the development of this software is the lack of features that would help to experiment, and as a result, understand the behavior of force-directed algorithms in the existing graph drawing software. Some of these features include smooth node-based and iteration-based animations, display of running-time and iteration counts, and variety of initial layout algorithms. LayoutShow supports a number of force-directed graph drawing algorithms as well as layouts based on eigenvectors. Node-based and iteration-based animations have been implemented. In addition, the software provides some algorithms for producing non-random initial layouts for force-directed algorithms. File I/O using GML file format has been implemented. Furthermore, users of LayoutShow applet can choose to perform local file I/O since LayoutShow is a signed applet. To our knowledge, LayoutShow is the first graph drawing software with this feature.

1 Introduction

At the start of our research on improving the existing force-directed graph layout algorithms which resulted in CostSpring layout algorithm [1], we realized that the existing graph embedding software do not accommodate all the features that are useful in experimenting with and better understanding the force directed algorithms. As a result, LayoutShow, an application/applet for graph drawing, was developed.

In this Paper, we first discuss LayoutShow's features. Then, we describe the way that different components of the system interact with each other. This is followed by two snapshots of LayoutShow. Finally, we present a list of known bug and limitations.

LayoutShow has been implemented using JDK 1.1.6 [13] under Solaris 2.5.1 (SunOS 5.5.1). It has also been tested in Linux 2.0.36, Windows 98, and Windows NT 4.0 as an application as well as an applet. The source, and bytecode of

J. Kratochvíl (Ed.): GD'99, LNCS 1731, pp. 242–249, 1999.

LayoutShow classes for down-load, and the LayoutShow applet can currently be found at "http://www.cs.yorku.cs/~lila/work.html".

2 Features

2.1 Outline

LayoutShow provides functionality to

- Generate a variety of graphs: complete graphs, rectangular , hexagonal, and triangular grids, complete binary trees, random graphs, hypercubes, and circular graphs.
- Draw graphs using different force-directed Spring algorithms: CostSpring[1] [1], GEM [4], FR [5], KK [8], and a combination of eigenvectors layout [11] and CostSpring [1].
- Generate initial layouts for Spring algorithms using: CostChosen [2] [1], Insert [4], and EigenLayout [11].
- Randomize a graph.
- Obtain graph quality measurements[3]: number of edge crossings, longest edge to shortest edge ratio, edge length deviation, and cost value: a graph layout quality measurement described in [1].
- Perform iteration-based and node-based animations. Some of the Spring algorithms find the forces acting on a node and reposition this node. In this case, it is possible to update the layout after a node is repositioned, node-based animation, or after all the nodes are repositioned once, iteration-based animation. LayoutShow gives the user the opportunity to choose one of these animation types, or no animation at all. For Spring algorithms such as FR [5] which move the nodes at once, only iteration-based animation is applicable.
- Configure the algorithms.
- Label the nodes.
- Reading/Writing a graph from/to disk using the GML [6] file format.

2.2 The LayoutShow Applet

We mentioned that LayoutShow is an application and also an applet. When running LayoutShow as an applet, all the classes that LayoutShow needs are down-loaded in one HTTP transaction. This has been achieved through the use of a JAR, Java Archive file [3]. This may slightly slow down the speed at which the applet is loaded, but it will certainly speeds up the execution once the applet

[1] CostSpring is a variation of spring-based graph drawing algorithms that has been developed as a part of the author's Master Thesis.

[2] CostChosen is an algorithm to generate initial layouts for force-directed algorithms that has been developed as a part of the author's Master Thesis.

[3] These quality measurements are mainly relevant to the layouts produced by a force-directed spring-based algorithm which LayoutShow focuses on.

is loaded and is running. A JAR file can also be compressed, as the JAR file for LayoutShow is. The current size of this file is about 0.5 megabytes.

Due to Java security restrictions a regular applet cannot access files on the local disk[4]. For this reason, VGJ [10], another graph drawing tool that is available as an applet, does not allow load and save operations. Although this restriction exists for applets by default, but Java has provided ways for permitting an applet to access a local disk and have all the privileges that a Java application has. This is done using *signing and verifying* JAR files. We have signed the JAR file of LayoutShow, providing the option of using the LayoutShow applet with all the privileges of an application including File I/O. This is particularly important since traditionally graph drawing applets such as VGJ did not allow file I/O that is an important part of graph visualization[5]. GDS [2] requires the user to send his/her data files to their server, and they return the URL of the file that contains the layout graph (produced on the server side) to the user. This approach is slow specially if the graph is large. In addition, with the increasing speed of the processors that average users currently have, there is no need for relying on a server to do the computations when the computation can be done locally and reasonably fast. The details of the concepts and procedures of signing and verifying an applet can be found in [3].

3 The Overall Design of the System

In this Section we elaborate on the way that various components of the LayoutShow software interact with each other to support the computations for finding the new node positions, drawing the graph, and animation. The computation of node positions for a graph (computation module), and its actual drawing on the screen (display module) make up the two main modules of any graph drawing software. Traditionally in software such as Graphlet [7] and VGJ [10], first the new positions of the nodes are computed, and then the graph is (re)drawn. These two actions happen in a sequential order. We have used the multi-threading capabilities of Java for simultaneous position computations and drawing of a graph. To the best of our knowledge on the Java-based graph drawing software, this is a new design on the way these two modules cooperate in such tools. Figure 1 show the relationship between the computation and the display modules.

The Two Threads The tasks of computation and display are managed by two threads: `drawThread` and `computeThread` that share the graph as a common data. As a result, the segments of the code in which these two threads read or modify the graph are considered as critical sections, and must not be executed

[4] Any discussion on Java security in this thesis refers to the Java 1.1.x security model [3], [13].

[5] This is because data may have been produced by another tool and saved in a file, and now the user needs to visualize this data.

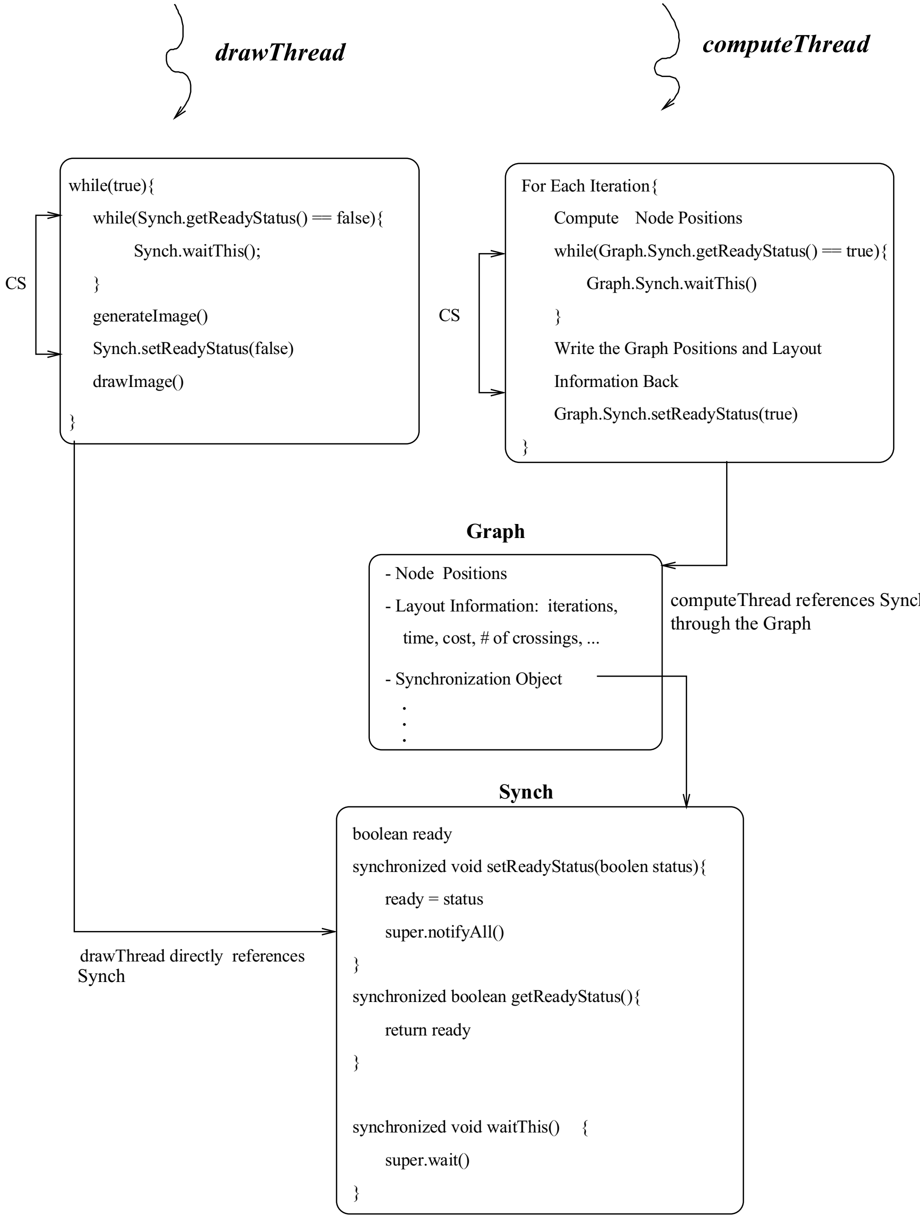


Fig. 1. Synchronization scheme between `drawThread` and `computeThread` in LayoutShow.

simultaneously[6]. Furthermore, the order in which these sections are executed is crucial.

[6] We must note that by simultaneous execution of threads we do not mean that threads run on multiple processors. What we refer to is the processor sharing by multiple threads through pre-emption.

The Critical Sections The two critical sections are:

1. The writing back of the new node positions by the `computeThread`.
2. The reading of these new positions by the `drawThread`.

It is clear that item 1 above should be executed before item 2, and synchronization is required to manage this.

Synchronization The synchronization of the critical sections are managed by using the wait and notify mechanism of Java [9] through a synchronization object: `Synch` (see Figure 1). As we can see in this Figure, the `drawThread` loops infinitely, and in each iteration if the Synch object indicates that the graph is not ready to be drawn then the `drawThread` waits on the `Synch` object[7]. On the other hand, after writing the node positions, the `computeThread` calls `Synch.setReadyStatus(true)` which in effect calls the `notifyAll` function of `Synch`. This resumes `drawThread` which now can start generating the image. At this point, the `computeThread` can continue computing the node positions for the next iteration. However, it cannot write the new positions back to the graph unless the `drawThread` has already called `Synch.setReadyStatus(false)`. If the `drawThread` has not called this function the `computeThread` will wait on the `Synch`. The `drawThread` makes this call after it reads the graph and generates the image, however, the actual drawing of the image occurs after the call to this function. As a result, the code segments that are labeled with CS (for critical section) in Figure 1 cannot be executed simultaneously by the two threads. And, the node positions are computed before the image is drawn. We must also note that the methods of `Synch` object are synchronized, and therefore, only one thread at the time may exist in this object.

For algorithm that don't have multiple iteration, or for when the animation option is off in force-directed algorithms, the node positions are only written back once at the end of the algorithm by the `computeThread`. However, in case of iteration-based animation the node positions are all written back once at the end of each global iteration, and the image is redrawn. In case of node-based animation, the position of the node that has moved is written back and the image is redrawn. For any choice of animation, the computation of the node position(s) and generation/drawing of the graph can occur concurrently resulting in a smoother animation.

Snapshots of LayoutShow Window

Figure 2 shows a snapshot of LayoutShow window with a hexegonal grid drawn using the CostSpring algorithm [1].

Figure 3 displays a snapshot of LayoutShow window performing the CostChosen initial layout algorithm [1]. The middle panel of main window of LayoutShow has `CardLayout` manager which allows for multiple `Component` objects to

[7] Object A waits on object B when object A calls the wait function of object B. A call to notify or notifyAll of B can resume A.

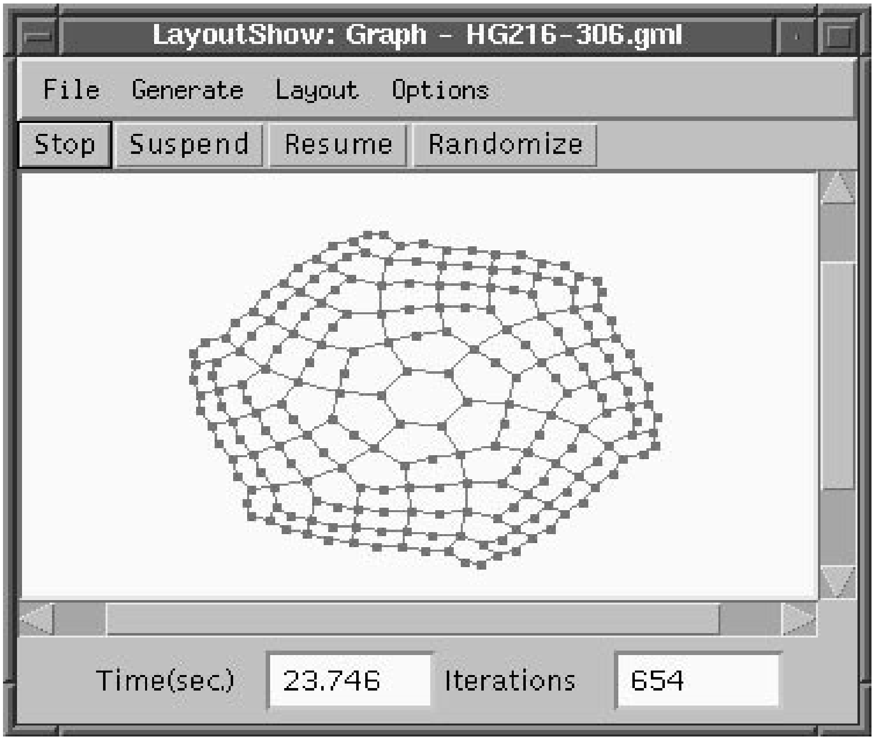


Fig. 2. Snapshot of LayoutShow's main window.

overlap [15]. This middle panel has two overlapping components: the default is a `Canvas` (see Figure 2), and the other one is a `Panel` with 10 `Canvas` objects (laid out in 5 columns and 2 rows) to support the simultaneous execution of Spring algorithms on different initial random layouts of a graph in CostChosen initial layout. The user can choose the number of initial layouts that are used where the maximum number of initial random layouts is 10. Once the algorithms for all initial layouts terminate, then the graph layout quality value of the resulting layouts will appear on top of each canvas with the lowest one flashing in red (see [1] for this quality value's formula). Then after a delay, the resulting layout with the lowest quality value will appear on the single canvas of main window.

4 Bugs and Limitations

The known bugs and limitations of the current implementation of LayoutShow are as follows

– LayoutShow currently does not provide graph editing facilities. This is a feature that will be added to the system.

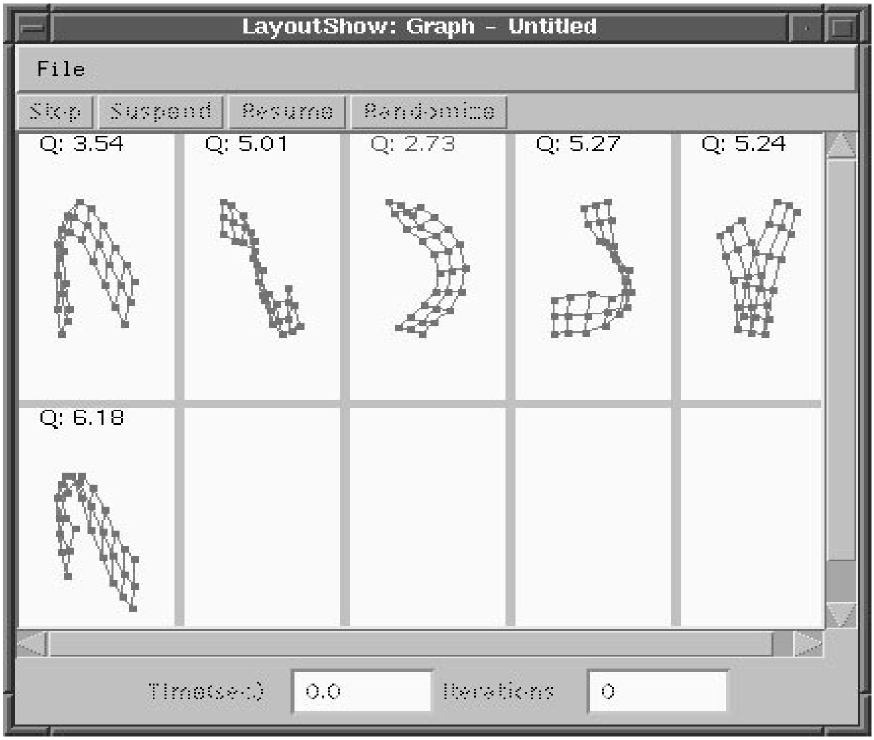


Fig. 3. Snapshot of LayoutShow's main window with a multiple-canvas panel.

- A very simple approach to labeling has been used which results in overlapping labels in some cases.
- Even though, our graph structure supports directed graphs, but the drawing of directed edges have not yet been implemented in LayoutShow.
- Currently, Netscape and Internet Explorer, the two commonly used Internet browsers, can only support Java signed applets [3] if they have a Java Plug-in [14] installed. Our LayoutShow applet is signed by JDK 1.1.6 [13], and tested using Java Plug-in 1.1 [12]. Although, Sun has promised that the final version of Java Plug-in 1.2 would also support the applets signed by JDK 1.1.x, but we have not tested our signed applet using Java Plug-in 1.2.
- The animation in a LayoutShow applet that is running under the default Java virtual machine of a Netscape browser sometimes hangs. We recommend using a Java Plug-in [12].

References

1. L. Behzadi A Cost-oriented Approach to Spring-based Graph Embedding in Layou tShow: a Java Environment for Graph Drawing. Master Thesis, York University, July 1999. Currently available at: `http://www.cs.yorku.cs/~lila/work.html`.
2. S. Bridgeman, A. Garg, and R. Tamassia. A graph drawing and translation service on the WWW. In *Proceedings of Graph Drawing '96*, pages 45–52. Springer-Verlag, 1997.
3. M. Campione, K. Walrath, and A. Huml. *The Java Tutorial Continued : The Rest of the Jdk.* Addison-Wesley, Massachusetts, December 1998.
4. A. Frick, A. Ludwig, and H. Mehldau. A fast adaptive layout algorithm for undirected graphs. In *Proceedings of Graph Drawing '94*, pages 388–403. Springer-Verlag, 1995.
5. T. Fruchterman and E. Reingold. Graph drawing by force-directed placement. *Software-Practice and Experience*, 21:1129–1164, 1991.
6. M. Himsolt. GML: A portable graph file format. Technical report, University of Passau, 94030 Passau, Germany, 1997. Currently available at: `http://www.fmi.uni-passau.de/Graphlet/GML/gml-tr.html`.
7. M. Himsolt. The Graphlet system. In *Proceedings of Graph Drawing '96*, pages 233–240. Springer-Verlag, 1997.
8. T. Kamada and S. Kawai. An algorithm for drawing general undirected graphs. *Information Processing Letters*, 31:7–15, 1989.
9. D. Lea. *Concurrent Programming in Java.* Addison-Wesley, Massachusetts, 1997.
10. C. McCreary and L. Barowski. VGJ: Visualizing graphs through java. In *Proceedings of Graph Drawing '98*, pages 454–455. Springer-Verlag, 1999.
11. T. Pisanski and J. Shawe-Taylor. Characterizing graph drawing with eigenvectors. Technical report, Royal Holloway, University of London, 1998. Currently available at: `http://www.ijp.si/tomo/papers/papers.htm`.
12. Sun Microsystems. *Java Plug-in Documentation*, 1998. Currently available at: `http://java.sun.com/products/plugin/1.1.2/docs/index.html`.
13. Sun Microsystems. *JDK 1.1 Documentation*, 1998. Currently available at: `http://java.sun.com/docs/index.html`.
14. Sun Microsystems. *Java Plug-in 1.2 Documentation*, 1999. Currently available at: `http://java.sun.com/products/plugin/1.2/docs/index.docs.html`.
15. J. Zukowski. *Java AWT Reference.* O'Reilly and Associates, California, 1997.

Centrality in Policy Network Drawings
Extended Abstract

Ulrik Brandes[1], Patrick Kenis[2], and Dorothea Wagner[1]

[1] University of Konstanz, Faculty of Mathematics and Computer Science
{Ulrik.Brandes, Dorothea.Wagner}@uni-konstanz.de
[2] Free University, Faculty of Social and Cultural Sciences, Amsterdam
p.kenis@scw.vu.nl

Abstract. We report on first results of a cooperation aiming at the usage of graph drawing techniques to convey domain-specific information contained in policy or, more general, social networks.
Policy network analysis is an approach to study policy making processes, structures and outcomes, thereby concentrating on the analysis of relations between policy actors. An important operational concept for the analysis of policy networks is centrality, i.e. the distinction of actors according to their importance in a relational structure. Matching structural with geometric centrality we incorporate the aggregated values of centrality measures into a layout model of the network.

1 Introduction

Given a graph from some area of application, utilization of general graph layout methods often yields drawings that are readable, but typically fail to communicate the important domain-specific information represented in the graph. Depending on the background, the same structure can clearly mean very different things, and different layouts can lead to different interpretations. To adhere to a fundamental principle of graphical excellence, that is to tell the truth about the data [16, p. 51], substance must therefore be considered already in the layout model.

We here report on a case study in which prominent domain-specific information contained in graphs from a sub-discipline of the social sciences is identified, and known layout models designed for readability of abstract graphs are adapted to account for substance. Complementary rendering rules are devised to contextually display further relevant information.

The case study presented originates from the field of public policy analysis. In the last decade, the policy network approach has become particularly prominent in the analysis of public policies. In contrast to other policy analysis approaches, the policy network approach has a much more realistic perspective of how policies develop, and considers policies principally as a result of a collaboration of a differentiated set of actors (public and private, local, regional and national, etc.). A policy network approach conceptualizes policy-making as the result of interactions between policy-actors, and assumes that the structure of

J. Kratochvíl (Ed.): GD'99, LNCS 1731, pp. 250–258, 1999.

these interactions explains policy outcomes [9]. A large number of structural characteristics is taken into account in the analysis and explanation of policy networks, but the most prominent one is certainly the notion of centrality. The reason being that public policy analysis is a sub-discipline of political science in which traditionally the principal question has been: “who has the power?”. Centrality is considered a fairly good indicator for power in networks, and formalizations of the centrality concept are typically based on degree, shortest paths, or eigenvectors of graph related matrices [17].

Except for an isolated historical example [13], we know of no attempt to visualize centrality, or, for that matter, any other structural variable of a network, and the underlying network at the same time. This is despite the fact that empirical studies stress the effect of layout on the perception of structural characteristics [11]. Note that structural centrality has an immediate geometric pendant. Though layout methods used for social networks in general, like multidimensional scaling and spring embedder variants [10,1], occasionally do a fairly good job of placing central actors close to the center, they are by no means reliable in this respect, but often misleading. We here show how to exactly represent the aggregated values together with the underlying network data.

In Sect. 2, we provide some background on the area of application, and develop a layout model and rendering rules in Sect. 3. Suitability of the resulting drawing model was established in an exploratory data analysis from which examples are presented in Sect. 4.

2 Policy Networks and Centrality

Centrality is considered a fundamental characteristic of policy networks since it gives an indication of the most important, the important and the unimportant actors in the network. The question is particularly relevant for mainly two reasons: first it tells us something about the social or political structure of policy making and secondly it helps understanding the outcomes that policy networks produce. The social or political structure of a network indicates which type of actor is involved in which way in the policy-making process. Who has access and control over resources and who has a brokerage position? From this perspective it makes considerable difference whether there is a most important actor in a network and, if this is the case, what type of actor it is. In terms of legitimacy, accountability, justice, etc. it makes a difference whether in the health policy field, for example, the most important actor is a state or a private actor and how important the actor is relative to the others. Moreover, there is evidence that the centrality structure of the network explains why a network was particularly successful in producing certain outcomes, or why policies have failed to come about.

A number of approaches have been devised to operationalize “importance”, all of which are equally accepted because they address different dimensions of the intuitive notion (for an overview see [6,7]). We limit our exposition to three exemplary measures that are used widely: degree, closeness, and betweenness centrality.

The *degree centrality*, $C_D(v)$, of a vertex v is simply the degree of that vertex, the idea being that the degree to which actors are active in relating to other actors is relevant. The actors with a high degree centrality are where the action in the network is. In policy networks these actors are highly visible for the other actors and are recognized by the others as a major channel of relational information.

This very local measure can be extended in several ways to take into account the whole graph. One is by taking the *closeness*, i.e. the sum of the distances to all other vertices, as a basis. *Closeness centrality*, defined as the inverse closeness, $C_C(v) = \left(\sum_{u \in V} d_G(u,v)\right)^{-1}$, focuses on how close an actor is to all the other actors in the network. If we consider a policy network where a certain actor has information which is crucial to all other actors one would expect this actor to have a high closeness centrality for the network to function effectively.

Finally, *betweenness centrality* is defined as the sum of the ratios of shortest paths between other actors that an actor sits on, $C_B(v) = \sum_{u \neq v \neq w} |P_v(u,w)|/|P(u,w)|$, where $P(u,w)$ and $P_v(u,w)$ are the sets of all shortest paths between vertices u and w, and those shortest paths passing through v, respectively. It indicates which other actors have control over the interaction between two non-adjacent actors. An actor with high betweenness centrality is between many actors in terms of shortest paths. In policy networks these actors are considered important because they control the spread of information between actors or sets of actors and thus can influence decision-making processes.

A framework to obtain normalized and network level centrality measures from a given actor level measure is described in [6]. Any centrality measure C is normalized to lie between zero and one by dividing its values by the maximum possible score in any graph with the same number of vertices. The above three measures thus yield *normalized measures* $C'_D(v) = \frac{C_D(v)}{n-1}$, $C'_C(v) = \frac{C_C(v)}{1/(n-1)}$, and $C'_B(v) = \frac{C_B(v)}{(n-1)(n-2)/2}$. Network *centralization* on the other hand quantifies the range of variability of the individual actor indices. A low network centralization in a policy network is thus an indication that there is not a clear center of action. Centralization is formalized as the cumulated differences between actor centralities and the maximum score attained in the present network, normalized by the maximum possible such sum. For all of the above measures, the star is a maximally centralized graph, whereas cliques and circles are not centralized at all.

The network in Fig. 1 shows that these measures actually differ. Each identifies a different set of maximally central vertices, marked by corresponding labels.

3 Centrality Drawings

Drawings of policy networks are to aid the exploration and communication of substantive content. Since centrality is such an important concept in policy network analysis, we want to devise means to make it visible in a graphical presentation of the policy network, rather than listing scores in a table (as is common).

Following [2], we think of graphical presentations as composed of graphical primitives, called *graphical features* (points, lines, areas, volumes), that represent data elements. These features have properties, called *graphical variables* (size, shape, color, etc.), that are either fixed according to a chosen form of representation, or varied according to the data. To produce an effective graph visualization, three main aspects have to be taken into account [3]:

- The graph's substance, i.e. the syntactic (intrinsic) and semantic (extrinsic) domain-specific information that is to be represented.
- The graphical design specifying which graphical features are to represent which data elements (*representation*), and how values shall be assigned to positional (*layout*) and retinal (*rendering*) graphical variables.
- The algorithm used to determine the layout, since many criteria of good graphical design are only approximately satisfiable and a particular algorithm may thus introduce particular artifacts.

Our representation of choice is the traditional *sociogram* [12]. In an attempt to make substance visible, our layout model maps structural to geometric centrality. We thus constrain each vertex to lie on a circle centered at the center of the diagram. The radius $r(v)$ of the circle for vertex v is determined from the structural centrality $C(v)$ of v. After experimenting with several other mappings, we decided to use radii

$$r(v) = 1 - \frac{C'(v) - \min_{u \in V} C'(u)}{\max_{u \in V} C'(v) - \min_{u \in V} C'(u) + c(G)}$$

where $c(G)$ is an offset used to avoid overlap if there is more than one vertex of maximum centrality. Showing levels as thin circles allows to compare centrality scores exactly, so that an accompanying tabular presentation is no longer needed.

The remaining degrees of freedom in the layout model should be used to ensure readability of the diagram. Unfortunately, the barycentric technique used for ring diagrams of hierarchies [14] is not suited for our problem, because transitive edges (not present in a hierarchy) may result in edge overlap.

The circular layering is also reminiscent of layered drawings of directed graphs (as in [15]), and a small number of edge crossing is a natural requirement is both cases. However, the number of crossings even between two centrality levels in a radial layout is not uniquely determined by the cyclic orderings of vertices on these levels. This observation still holds, if positions in one layer are fixed. By necessity, radial differences between centrality levels are fixed. Note that this is different from the usual approach to layered drawings of directed graphs, where layer distances are computed only after determining the orderings of vertices on each level. Fixed level differences suggest to impose the additional constraint that edges between two levels may not pass through the inner circle (outward drawing). Crossing minimization is then easily reduced to two-layer crossing minimization with one fixed layer, which is $\mathcal{NP}$-complete [5]. Moreover, bend edges (necessarily introduced by two-layer methods) tend to be rather confusing in radial layouts. We thus decided not to try adaptations of methods devised for layered layouts.

To try out the effectiveness of different designs, we instead exploited the flexibility of energy-based placement approaches. Our prevailing objective function uses the pairwise spring potentials of [8], where preferred distances are determined from shortest paths in the underlying graph, where edge lengths are redefined according to level span and number of actors on similar levels. Additionally, we used vertex-edge repulsion and crossing counts similar to those in [4]. To achieve better overall organization, the layout procedure employs three stages working on different subgraphs, and the various penalties are dependent on the annealing temperature. Details appear in the full paper.

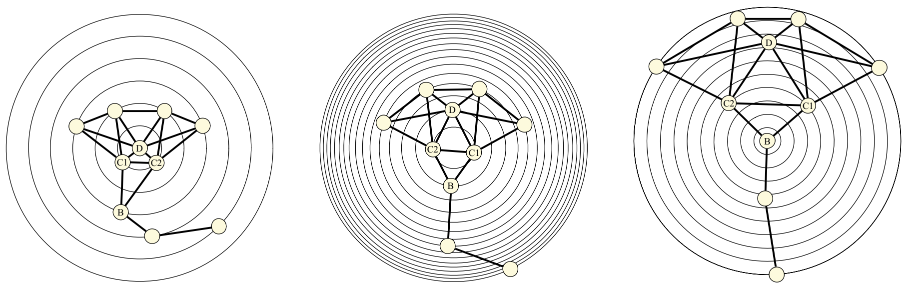


Fig. 1. Layouts showing degree, closeness, and betweenness centrality, respectively

While the layout is designed to organize the network in a readable way that exactly represents actor centralities, we can use the remaining graphical variables to convey further substance. Due to space limitations we briefly skim over them. In the examples below we have coded legal status and attitudes towards the issue by shape and color of vertices, respectively. Confirmed relations are drawn as thick black lines, whereas unconfirmed relations are drawn as grey lines with an arrow indicating who mentioned whom (see below). In the directed network of claimed relationships, the ratio of the in- and outdegree of each vertex measures the reciprocation of an actor's claims. We therefore vary the size of point features representing actors so that the ratio of height and width equals the ratio of in- and outdegree, while the area is proportional to the combined degree.

4 Analyzing Local Drug Policies

In this final part the usefulness of the graph drawing techniques presented above will be demonstrated. The demonstration is based on data from a project studying the incidence of HIV-preventive measures for IV-drug users in 9 German municipalities. The research question underlying the project is: why differ these municipalities so much in the provision of HIV-preventive measures (such as methadone substitution and needle exchange) given the fact that the problem load (i.e. the number of IV drug users and the HIV epidemiological situation) are very similar? The study tests the hypothesis whether the difference in the

provision in HIV-preventive measures can best be explained by the structure of the policy networks. Given the amount of controversy and complexity involved in the provision of such measures the hypotheses is that they are contingent on types of relations between the different policy actors. The policy networks studied here include all local organizations directly or indirectly involved in the provision of such measures.

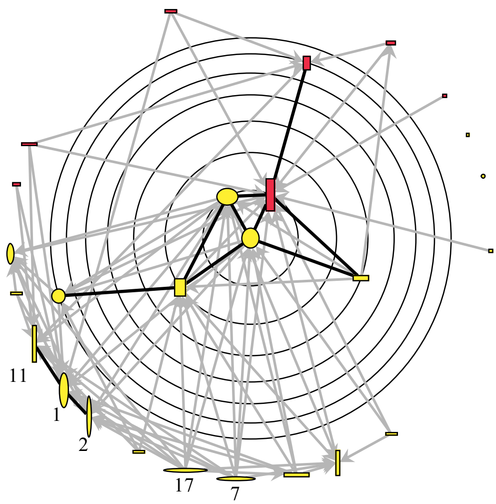


Fig. 2. Organizations involved in local drug policy making in Stuttgart

To sketch how a drawing can elucidate structural aspects of a network, we first take a look at the network of informal communication in the city of Stuttgart (Fig. 2). Informal communication is in principle an undirected relationship: if A talks to B, you assume that B also talks to A. What you see very often in empirical investigations, however, is that many undirected relationships are not reciprocated: although A mentioned B as a communication partner, B did not mention A. Consequently, policy network analysts often make a principal decision and either decide to calculate centrality on the basis of only confirmed or of both confirmed and unconfirmed relationships. In contrast, an analysis of this phenomenon on the basis of the drawing presented tells a number of additional stories. While the unconfirmed network appears fairly integrated, the confirmed network is sparse. The drawing however reveals that there is a crucial separation due to actors 1, 2, and 11, who are peripheral only because they seem reluctant to name others. Actors 7 and 17 (small non-profit organizations) are also interesting since they claim a huge number of relationships, none of which are reciprocated. Consequently, on the sole basis of these drawings a lot of additional facts are provided which help to come to an adequate evaluation of who actually the most important actor is.

As a second example we demonstrate the use of imaging in the analysis of the relationship between network structure and network outcome (i.e. in the present case the provision of HIV-preventive measures). The three networks presented in Fig. 3 (Essen, Köln, and Ahlen) differ a lot in their outcome effectiveness (Essen being the most and Ahlen being the least effective) but also differ a lot in their structural characteristics with respect to different measures of centrality.

On the basis of the drawings a number of additional relationships become visible which can be further developed regarding the effectiveness of networks: the number of rather active actors (i.e. with high degree centrality); the degree of discrepancy between confirmed and unconfirmed links; the fact whether a mixture of different types of actors (public and private, and repressive and supportive) are found close the center of the network; the fact whether there is one clearly central actor in the network, a couple of central actors, or no clearly central actors in the network. It is this type of inductive observations, which are a direct result of the graph drawing techniques and which contribute substantially to the analysis of the questions "who has the power?" and "what are the consequences of the power structure?".

Acknowledgment. We thank Vanessa Kääb for providing several earlier implementations of different centrality measures and experimental layout models.

References

1. Vladimir Batagelj and Andrej Mrvar. Pajek – Program for large network analysis. *Connections*, 21:47–57, 1998.
2. Jacques Bertin. *Semiology of Graphics: Diagrams, Networks, Maps.* University of Wisconsin Press, 1983.
3. Ulrik Brandes, Patrick Kenis, Jörg Raab, Volker Schneider, and Dorothea Wagner. Explorations into the visualization of policy networks. *Journal of Theoretical Politics*, 11(1):75–106, 1999.
4. Ron Davidson and David Harel. Drawing graphs nicely using simulated annealing. *ACM Transactions on Graphics*, 15(4):301–331, 1996.
5. Peter Eades and Nicholas C. Wormald. Edge crossings in drawings of bipartite graphs. *Algorithmica*, 11:379–403, 1994.
6. Linton C. Freeman. Centrality in social networks: Conceptual clarification. *Social Networks*, 1:215–239, 1979.
7. Noah E. Friedkin. Theoretical foundations for centrality measures. *American Journal of Sociology*, 96(6):1478–1504, May 1991.
8. Tomihisa Kamada and Satoru Kawai. An algorithm for drawing general undirected graphs. *Information Processing Letters*, 31:7–15, 1989.
9. Patrick Kenis and Volker Schneider. Policy networks and policy analysis: Scrutinizing a new analytical toolbox. In Bernd Marin and Renate Mayntz, editors, *Policy Networks: Empirical Evidence and Theoretical Considerations*, pages 25–59. Campus Verlag, 1991.
10. David Krackhardt, Jim Blythe, and Cathleen McGrath. KrackPlot 3.0: An improved network drawing program. *Connections*, 17(2):53–55, 1994.
11. Cathleen McGrath, Jim Blythe, and David Krackhardt. The effect of spatial arrangement on judgments and errors in interpreting graphs. *Social Networks*, 19(3):223–242, 1997.

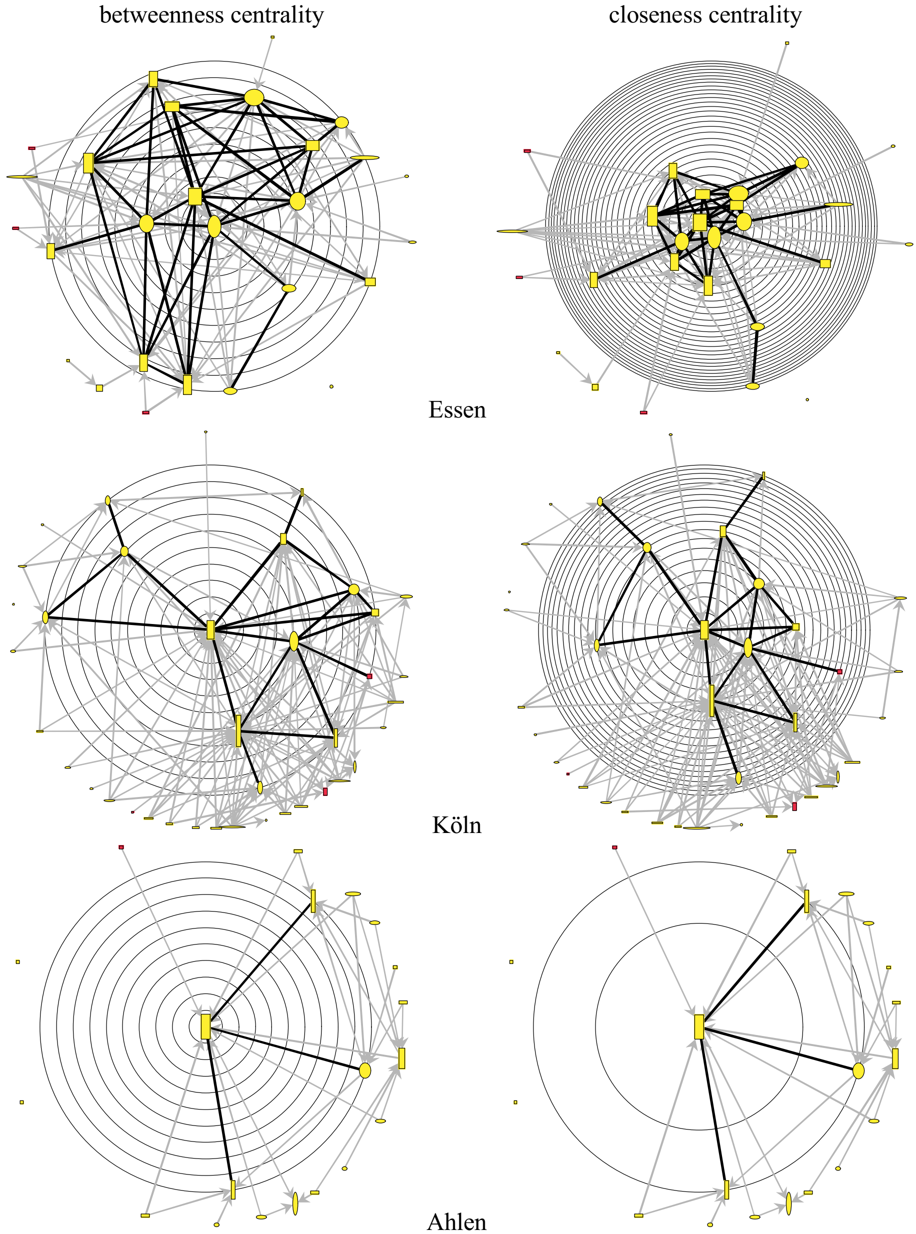


Fig. 3. Networks of informal communication. Centralities are according to confirmed networks, while vertex size codes normalized in- and outdegree centrality in unconfirmed networks and is comparable across drawings

12. Jakob L. Moreno. *Who Shall Survive: Foundations of Sociometry, Group Psychotherapy, and Sociodrama.* Beacon House, 1953.
13. Mary L. Northway. A method for depicting social relationships obtained by sociometric testing. *Sociometry*, 3:144–150, 1940.
14. Marcello G. Reggiani and Franco E. Marchetti. A proposed method for representing hierarchies. *IEEE Transactions on Systems, Man and Cybernetics*, 18(1):2–8, 1988.
15. Kozo Sugiyama, Shojiro Tagawa, and Mitsuhiko Toda. Methods for visual understanding of hierarchical system structures. *IEEE Transactions on Systems, Man and Cybernetics*, 11(2):109–125, February 1981.
16. Edward R. Tufte. *The Visual Display of Quantitative Information.* Graphics Press, 1983.
17. Stanley Wasserman and Katherine Faust. *Social Network Analysis: Methods and Applications.* Cambridge University Press, 1994.

Straight-Line Drawings of Protein Interactions
System demonstration

Wojciech Basalaj[1] and Karen Eilbeck[2]

[1] University of Cambridge Computer Laboratory
Pembroke Street
Cambridge CB2 3QG, UK
wb204@cl.cam.ac.uk
[2] University of Manchester Biochemistry Division
2.205 Stopford Building, Oxford Road
Manchester M13 9PT, UK
mobfeke1@fs2.scg.man.ac.uk

Abstract. This paper presents the first attempt at automatically generating drawings of protein interaction graphs. Such graphs are large and not necessarily connected. A straight-line drawing method based on the spring embedder metaphor has been found highly suitable for this task. The drawings produced exhibit edge length uniformity, even vertex distribution, and preserve graph topology well. This method is capable of generating both two- and three-dimensional layouts. A preliminary evaluation has been carried out based on a representative collection of interaction graphs.

1 Introduction

Protein-protein interaction data can be represented as a graph, with individual proteins taken as vertices, and the existence of an interaction between a pair of proteins denoted by an edge. These data are inherently complex, and graph drawing techniques can be used to convey the underlying structure effectively. Visual representation is an ideal method to communicate protein-protein interactions within the biological community.

This paper starts by giving the motivation for using visualisation techniques to analyse protein interaction data. Our approach is outlined and set in the context of related work. We then go on to discuss the details of the chosen graph drawing method, and show its suitability to the class of graphs typical of this domain. Our experiences with applying this method follow. We present insights contributed by domain experts when exploring protein interaction drawings. Finally, we speculate on future directions of our work, and offer some concluding remarks.

2 Biological Context

The genome projects such as Yeast [14], C. elegans [12], human [8], and E. coli [5], have produced a lot of biological data, pinpointing genes to places on the

J. Kratochvíl (Ed.): GD'99, LNCS 1731, pp. 259–266, 1999.

chromosomes, discovering analogies between species, locating disease genes. This data in itself is immense, but it is the post genome sequencing analysis that will produce vast amounts of complex experimental results. It is an attempt to make sense of the cells, tissues, organs and organisms that make use of the genetic load they are given. The analysis of protein-protein interactions endeavours to bring understanding to the function of both individual proteins and networks of proteins within the cellular context. An estimate is that about 40% of genes from sequencing projects are novel and have no assigned function [7]. Knowing the interactions that these proteins make can help to pinpoint the role that they play. The data is being gathered in both small scale experiments and large industrial size ventures, such as the plan to use the Yeast two hybrid method to map all of the interactions made by each of the 6000 proteins in yeast [13].

Currently the data from these experiments is reproduced mainly as tables of interactions, as displayed by the MIPS yeast genome resource centre [18]. Protein linkage maps have been produced for small genomes such as the bacteriophage T7 [2], but again this information is displayed in a tabular form. Specific databases such as Ecocyc, the E coli Metabolic Database, have represented pathways of interacting enzymes as automatically generated hierarchical diagrams [16].

The INTERACT database of protein-protein interactions [11] has been used as an information resource. The data held is of diverse interaction types: structural, enzymic, transient, permanent, therefore the corresponding graphs are undirected, unlike for the metabolic data. Two subsets of data have been used, Yeast, and E. coli, both fully sequenced unicellular organisms. The data from both of these subsets is of high cardinality and connectivity, so it is obvious that a pictorial representation via a straight-line drawing is best. Figures 3 and 4 are example visualisations of yeast interactions.

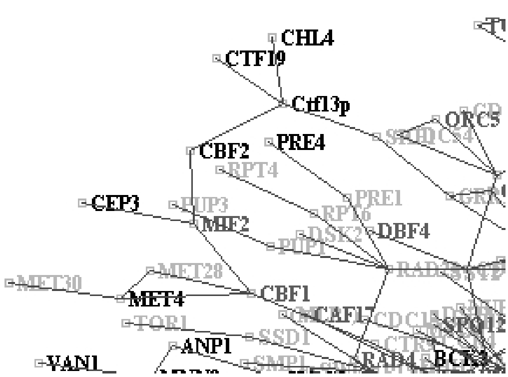


Fig. 1. Details of interactions in yeast with those proteins not present in C. elegans highlighted in black. The grey labelled proteins are only found once in C. elegans, whereas the light grey proteins have been replicated several times in the new genome

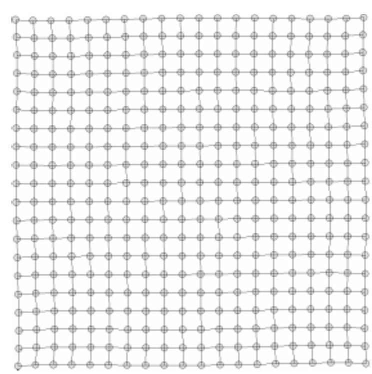


Fig. 2. 2D drawing of 20x20 grid

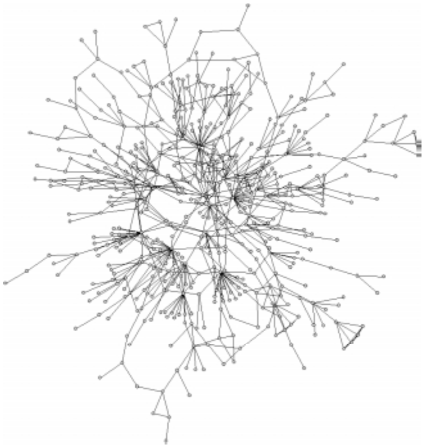


Fig. 3. Main component of interactions in yeast represented as a straight-line drawing in 2D. There are 525 vertices and 724 edges in this graph

Fig. 4. 3D representation of interactions in yeast. There are 766 vertices and 924 edges in this graph

3 Large Straight-Line Drawings

Traditional aesthetics for straight-line drawings of general undirected graphs, such as display symmetry, avoiding edge crossings, and uniform edge lengths [4], are in strong conflict for large graphs. Fortunately, this is relatively unimportant, since actual edge connections are perceptible only on a local scale. On the whole, the location of vertices gains more significance as it becomes the dominant factor for judging graph-theoretic distances. Therefore, if a proper overview of a graph is to be achieved these distances should be approximated by Euclidean distances between vertices in a drawing.

A function, that for a given pair of vertices returns the length of the shortest path between them, satisfies metric properties:

- $f(a, b) = f(b, a)$, true for an undirected graph
- $f(a, b) \geq 0$, with $f(a, b) = 0$, if and only if $a = b$
- $f(a, b) \leq f(a, c) + f(c, b)$, for all vertices a, b, c. If c is a vertex on the shortest path between a and b then $f(a, b) = f(a, c) + f(c, b)$. Otherwise, $f(a, b) > f(a, c) + f(c, b)$ cannot hold because a path through c would be shorter.

These properties are preserved for a disconnected graph if the length of the shortest path between vertices from separate components is considered to be infinite (although, for practical purposes a value greater than the longest path in all components will suffice). This function is a dissimilarity metric for graphs.

Multidimensional Scaling (MDS) is a standard multivariate analysis method [9], that can be applied to a given dissimilarity metric and a data collection, to yield a matching configuration of points, with each point representing a single

data object. Least-squares metric MDS uses a numerical minimisation process to ensure that Euclidean distances in the configuration approximate dissimilarities between corresponding data objects. For the graph metric this will give the desired result of approximating graph-theoretic distances. This approach to graph drawing was first adopted by Kruskal and Seery [17].

Incremental MDS [3] is a least-squares metric MDS algorithm that is capable of laying out even very large data collections (100,000 objects) in a satisfactory time on a standard workstation. This performance is achieved at the slight expense of layout quality - details of this mechanism can be found in the original paper. For smaller data collections (up to thousands of objects) it is feasible to resort to the standard, non-incremental mode. At this level the algorithm is essentially equivalent to that of Kamada and Kawai [15]. It emulates a fully connected spring system with one anchor point for every vertex. The relaxed length of a spring connecting two points is taken to be the dissimilarity between the corresponding pair of vertices (shortest path length). The actual length of the spring is the Euclidean distance between its two anchor points. The algorithm attempts to arrive at a minimum energy state (an optimal configuration) by reducing the disparity between actual and desired lengths. This process is iterative and terminates when the amount of improvement becomes negligible.

The energy potential of the spring system is given by a loss function

$$E = \sum_{r<s} \frac{\left(d_{rs} - \hat{d}_{rs}\right)^2}{\hat{d}_{rs}^2} \tag{1}$$

that is being minimised. In this formula d_{rs} denotes the actual length of the spring connecting points r and s, $\hat{d}_{rs}$ is the relaxed length of the spring. Since contributions of individual springs are normalised, any deviation in distance between adjacent vertices is penalised more than the same error for non-adjacent vertices. Therefore, a straight-line drawing based on an optimal vertex layout will have uniform edge lengths. At the same time an even spread of vertices is achieved, because distances between non-adjacent vertices tend to be proportional to their shortest path lengths. These properties are illustrated by a graph drawing in Figure 3. This algorithm does not explicitly minimise edge crossings or maximise drawing symmetry, however one or both of these aesthetics might be correlated with the optimisation criterion of preserving graph-theoretic distances, for certain graphs. An optimal layout of a 20x20 grid in Figure 2 serves as an example.

MDS can be used to embed a graph in an Euclidean space of any dimension. 2D layouts are an obvious choice, however 3D drawings can be advantageous, particularly for strongly connected graphs: in three dimensions edge crossings are less likely to occur. Also, graphs with a large number of disconnected components can benefit from a three dimensional representation. Please note how small components are arranged on a hemisphere around the main component in Figure 4. This arrangement will be squashed to a semicircle in two dimensions.

For converting an MDS layout to a 2D drawing we have created a simple viewer. It has facilities for labelling vertices, colouring vertices and edges, and

storing the drawing as an image file. Figures 1 and 3 were created with this program. Interactive examination is crucial for gaining an understanding of an abstract 3D representation. A combination of a 3D scene description language and a viewer achieves this goal in a flexible way. We have decided to use Virtual Reality Modelling Language (VRML) [6], which can be viewed in a web browser with an appropriate plug-in or a stand-alone VRML viewer. This results in a very portable method of disseminating these drawings. Figures 4-7 are snapshots taken along a sample navigation path.

Andalman et al have explored the use of MDS for creating VRML visualisations of synthetic image collections - Design Galleries [1]. A dissimilarity metric capturing perceptual distance between images has been used. An image collection can be seen to be equivalent to a complete weighted graph, with images as vertices, and dissimilarities between pairs of images as edge weights. A visualisation consists of images arranged at the coordinates calculated by MDS. To avoid clutter, edges are omitted as their weights are approximated by image distances.

4 Evaluation

It is customary for protein interaction graphs to be drawn manually or presented in a tabular form. This approach is not suitable for large data sets, however. To our knowledge, it is the first time an automatic graph drawing technique has been applied in this context. The visualisation of all of the proteins and interactions between them has provided us with a new means in which to study the data stored.

Upon inspection of the clusters in the VRML representation of yeast interactions (Figure 4 represents a starting point of exploration), it became apparent that there are several errors in the database. These errors were visible to a 'yeast' domain expert who recognised that some patterns were incorrect. In the first instance, the protein ALG5 is shown in Figure 5 to interact with many proteins,

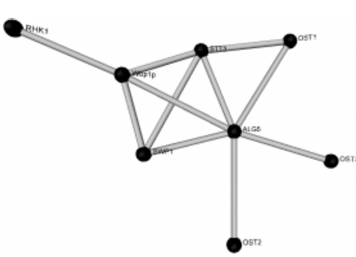


Fig. 5. Neighbourhood of ALG5 protein

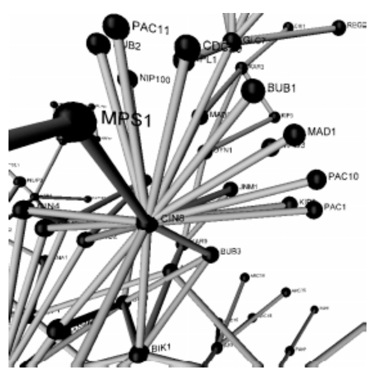


Fig. 6. Neighbourhood of CIN8 protein

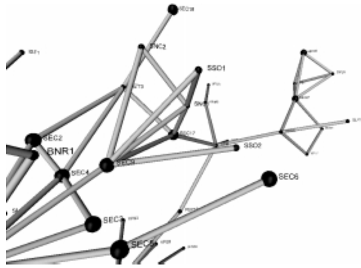


Fig. 7. Neighbourhood of SEC proteins. Spheres representing proteins of interest have larger diameter

when in fact it interacts with only one of the displayed proteins, WBP1. In the second instance, it was observed that the protein CIN8 exhibited far more interactions than actually exist, as seen in Figure 6. Both of these errors were undetected until the interactions were visualised. They have been attributed to experiments relying on genetic methods, which can sometimes produce false positive results.

The location of proteins within the graph drawing appears to be logical. For example, the SEC proteins, responsible for vesicle mediated transport, are located together in Figure 7; proteins involved in signal transduction are also grouped. The aspect of the layout that concerned the biological end user, was the question of whether vertex proximity in the drawing equated to close cellular location of the corresponding proteins. This is not true for cases where the proteins in question do not directly interact; however such proteins are unlikely to be close in the drawing anyway, as they are not adjacent in the graph. Another related aspect, that was pleasing to the user, was the scarcity of overly long links between pairs of proteins, that cross over many other links.

When analysing the protein interaction graphs, the end users found it most productive to view both the 2D and 3D drawings synchronously. It is possible to lose context and become 'lost in space' while navigating a 3D representation, so the 2D diagram is useful for regaining orientation. Using this approach it is then possible to study the more complex regions of nodes with many spokes like SNP1, and very highly connected complexes such as the NUP complex.

The data represented is highly connected, and the complexity can make certain regions difficult to comprehend. Filters were used to restrict the amount of data displayed. This came in the form of how believable the interaction actually is. Each interaction was assigned a number depending on the confidence in the experimentation. A series of graphs were then created from the skeleton of definite interactions, with the layers of less certain interactions added in different colours. This technique allowed the biologists to view the data in a simple form and then increase the complexity to add to their understanding, while preserving their mental map between stages [10].

Another application of graph drawing has been the representation of comparative studies between organisms. The 2D drawing in Figure 1 has been colour

coded to display a comparison between the Eukaryotes: yeast and C. elegans. The proteins responsible for making yeast a single cellular organism are obviously not present in C. elegans (a multi-cellular nematode worm), so are coloured differently. This diagram highlights the protein interactions of the kinetochore complex, which has a mitotic function in yeast. This representation has allowed the rapid identification of pathways and complexes unique to yeast, and also conserved eukaryotic interactions. Comparative analysis is useful as it provides a mechanism to pinpoint the function of proteins. (Eukaryotes are organisms with complex compartmentalised cells, ranging from single cell lower eukaryotes such as yeast to multicellular higher eukaryotes which are anything more complex than sponges. The other two domains of life are Eubacteria and Archaea, which both encompass single-celled life.)

5 Further Work

3D graph drawings in VRML are very interactive, and encourage exploration of the underlying data. On the other hand, static 2D drawings that we have produced so far are of limited usefulness. They serve as an overview of the graph structure, but fail to convey detailed relationships. To alleviate this situation a zoom facility is needed. Fisheye magnification seems most appropriate to use, as it preserves context. The SHriMP layout adjustment algorithm [19] avoids excessive distortion to peripheral objects, and we shall adopt it in the future.

The advantage of using MDS for calculating layouts of protein interactions is that it is not specific to a graph representation of these data. Instead of a dissimilarity metric based on graph-theoretic distances, we can use one that takes the location of proteins within the cell into account. A comparison of layouts resulting from both data representations might help to clarify the relationship between the location of a protein and its interactions.

6 Conclusions

A combination of MDS and a graph-theoretic metric appears to constitute an ideal graph drawing technique for protein interaction graphs, and others of similarly high level of complexity. The drawings produced exhibit aesthetics typical to that of spring embedder algorithms, and additionally preserve the underlying graph topology.

Automatically generated 2D and 3D graph drawings are a welcome alternative to existing methods of analysing protein interactions. They impose a tangible representation, to which users can readily relate, upon these complex data. These visualisations are invaluable for discovering data inconsistencies, and carrying out comparative studies.

7 Acknowledgements

Wojciech Basalaj's work is supported by Trinity College, Cambridge, and by the Overseas Research Students Awards Scheme. Karen Eilbeck's work is supported by a BBSRC case studentship with GlaxoWellcome. Thanks to Dr Ian Dix, for his yeast protein domain expertise.

References

[1] B. Andalman et al. Design gallery browsers based on 2D and 3D graph drawing (demo). In *Proc. of Graph Drawing '97*, volume LNCS 1353, pages 322–329, September 1997.
[2] P. L. Bartels et al. A protein linkage map of Escherichia coli bacteriophage -T7. *Nature Genetics*, 12:72–77, 1996.
[3] W. Basalaj. Incremental multidimensional scaling method for database visualization. In *Proc. of Visual Data Exploration and Analysis VI*, volume SPIE 3643, pages 149–158, January 1999.
[4] G. Di Battista et al. Algorithms for drawing graphs: an annotated bibliography. *Computational Geometry: Theory and Applications*, 4:235–282, 1994.
[5] F. R. Blattner et al. The complete genome sequence of Escherichia coli K-12. *Science*, 277:1453–74, 1997.
[6] R. Carey and G. Bell. *The Annotated VRML 2.0 Reference Manual.* Addison-Wesley Developers Press, Reading, MA, USA, 1997.
[7] R. A. Clayton et al. The first genome from the third domain of life. *Nature*, 387:459–462, 1997.
[8] F. S. Collins et al. New goals for the U.S. Human Genome Project: 1998-2003. *Science*, 282:682–9, 1998.
[9] T. F. Cox and M. A. A. Cox. *Multidimensional Scaling.* Chapman & Hall, London, 1994.
[10] P. Eades et al. Preserving the mental map of a diagram. In *COMPUGRAPHICS '91*, pages 34–43, September 1991.
[11] K. Eilbeck et al. INTERACT: An object oriented protein-protein interaction database. In *Proc. of ISMB (in press)*, 1999.
[12] The C. elegans Sequencing Consortium. Genome sequence of the nematode C. elegans: a platform for investigating biology. *Science*, 282:2012–8, 1998.
[13] M. Fromont-Racine, J.C. Rain, and P. Legrain. Toward a functional analysis of the yeast genome through exhaustive two-hybrid screens. *Nature Genetics*, 16:277–82, 1997.
[14] A. Goffeau et al. Life with 6000 genes. *Science*, 274:563–7, 1996.
[15] T. Kamada and S. Kawai. An algorithm for drawing general undirected graphs. *Information Processing Letters*, 31:7–15, 1989.
[16] P. D. Karp et al. Ecocyc: Encyclopedia of Escherichia coli genes and metabolism. *Nucleic Acids Res*, 27:55, 1999.
[17] J. B. Kruskal and J. B. Seery. Designing network diagrams. In *Proc. of the First General Conference on Social Graphics*, pages 22–50, October 1978.
[18] H. W. Mewes et al. MIPS: a database for genomes and protein sequences. *Nucleic Acids Res*, 27:44–8, 1999.
[19] M.-A. D. Storey and H. Mueller. Graph layout adjustment strategies. In *Proc. of Graph Drawing '95*, volume LNCS 1027, pages 487–499, September 1995.

Art of Drawing
(Towards Technology of Art)

Jaroslav Nešetřil

Charles University
Malostranské nám. 25
11800 Praha 1, The Czech Republic
nesetril@kam.ms.mff.cuni.cz

To Jiří Načeradský on the occasion of his 60ies birthday.

Abstract. This is an extended summary of an invited talk given by the author at Graph Drawing 1999. The text contains some of the basic ideas together with an outline of the whole lecture. As an appendix we included Nešetřil – Načeradský 1995 Manifesto.
This paper does not contain any of the extensive illustrations which accompanied the lecture (of course this is a paradox in itself: a paper on drawing without drawings). This is necessary due to the format of these proceedings. Moreover the lecture was conceived as a multimedia show (with slides, CD projection and transparencies accompanying the lecture on three different screens) and this is hard in any case to reproduce in the print form.
Thus, with a single exception, there are no accompanying figures here. For the benefit of the reader we included the list of our exhibition and an interested reader can get more complete information from the preprint KAM-DIMATIA Series 99-437 ([10]). One can also consult the two-volume set [11] and related works of the author [8], [9] and [5] (and references given there). This text was prepared especially for the conference on Graph Drawing.

1 Introduction

This text is based on an invited talk given by the author at GD'99. The text contains some of the basic ideas together with outline of the whole lecture.

This lecture was conceived as a multimedia show with slides, transparencies and CD projection (thanks to Hubert de Fraysseix) accompanying the lecture on three different screens called: Samples, Stories and Souvenirs.

All the illustrations on Samples and Souvenirs were related to ongoing artistic collaboration of Jiří Načeradský and the author. Jiří Načeradský (born 1939) is the foremost Czech artist who distinguished himself very early (e.g. new figuration) in the sixties and after 1989 was a professor at both the Academies of Art in Prague and Brno. He is represented in many major public and private collections locally and abroad.

J. Kratochvíl (Ed.): GD'99, LNCS 1731, pp. 267–275, 1999.

Our collaboration started more than 10 years ago and developed from discussions and exchanges of ideas to joint projects and to actual collaboration on canvases and other media (which we sign jointly). Our list of exhibitions includes: VIA Art , Prague 1995, House of Arts, České Budějovice, 1997, Karolinum and Malostranská beseda, Prague, 1998, Rabasova Galerie, Rakovník 1998-9.

In this paper we want to give an overview of the verbal part of the lecture (as projected on Stories) together with some of the basic ideas together with an outline of the whole lecture. However this paper does not contain any of the extensive illustrations which accompanied the lecture. To preserve at least some of the authenticity we continue this text in ‘Ich form’. Some comments about missing illustrations are in [ ]-brackets.

2 Drawing and Sketching

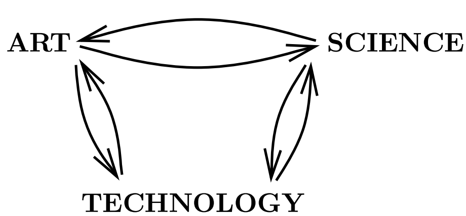
This is a volume on Graph Drawing. Thus I felt that it is perhaps fitting to speak about art here, and I accepted the invitation of the organizers. Graph Drawing is an area with Kandinski algorithms and even a Gioto algorithm (in both cases just a coincidence and a ‘poetic’ terminology). Perhaps more importantly this area involves aesthetic criteria which are explicitly discussed (also in several papers in this volume). It is my conviction that matters which we are going to touch upon are deep and profound. To reformulate the words of Kandinsky [6] (which he used when comparing art and music):

> In our opinion the similarities of Art and Mathematics are evident but they lie very deep.

I also like the opening lines of the scholarly work of H. Damisch [1] where he motivated his work with:

> Unpatience how the problems of perspective are treated without a deeper philosophical context.

2.1 The Basic Scheme



This scheme has the following features which I elaborated in the talk:

historical numerous evidence for the existence of all arrows from early history;

contemporary think of contemporary *international style* of installations and media related action art so well documented by 1999 Bienale di Venezia;

key words these include: methodology, honesty, drive for new, respect to the tradition,... versus freedom, aesthetic, public attention, market influence,...

In the lecture I illustrated mostly the relevance of the following arrows:

$$\text{ART} \longrightarrow \text{SCIENCE}$$
$$\text{ART} \longrightarrow \text{TECHNOLOGY}$$
$$\text{SCIENCE} \longrightarrow \text{ART}$$

Here I concentrate on the first interaction item only.

Where do we see the best evidence of these interactions, of such contacts?

As expected this is best documented on the remarks on frontiers of these activities, somewhere on the boundaries. Thus for the majority of my talk I decided to speak about sketches and about sketching as a typical drawing activity. I believe this is directly linked to some of the central problems of Graph Drawing.

2.2 Defining Sketches

Sketches are not related just to drawings. They are not restricted to the particular material used. Remember the great sketches of Rubens in oil (e.g. in Munich, in Prague) or sketches of sculptures which are necessary for major projects (DIMATIA houses a beautiful bronze statue - sketch of Charles IV for historical aula of Charles University by Jan Pokorný, 1947).

[I gave here further numerous examples, old and new.]

What then are sketches?

I believe they are defined by the *action* and not media, by their *context* and intended *purpose* (of their creation).

The action is usually fast and frankly mirrors experience and underlying skills; the context is usually important and meaningful and often leads to some of the key works of artists (and as a consequence many ‘unrelated’ sketches are destroyed as ‘meaningless’); and the purpose is as important as the context – sketches serve to free imagination, to test the unknown ground, to prepare and to organize thoughts and forms in relation to future projects.

I think in this sense one can consider sketches as *eternal anticipation* of action and conceptual art, or as a privatissimo of both, as a daring individual action in a world restricted by concepts, traditions and trends.

[This I documented thoroughly in the lecture.]

2.3 Nature of Sketches

The action- and conceptual- definition of sketches is not the only possibility. Another is their *freeness* and *gesture* which reflects and anticipates the mind, abilities, dreams.

They seem to be *encodings of fantasy.*

This striking freedom is surprising but it can be explained as a combination of several factors. I listed and documented several such aspects which make this freedom possible

contextual they are kind of privatissimo;
technological easy means to master;
physiological easily controllable small movements as opposed to large formats;
randomness unconsciousness, modern 'blow up' effect.

Yet these are just some side factors of some masterpieces which one can compare to little jewels (of mind and skill).

What a sovereignty and grandeur, what a style!

[Here were more examples spanning six centuries.]

The sketches also reveal very frankly the talent and the degree to which a particular artist had to *learn* the skills. Some seem to have a gift from God himself (say Raphael, Rembrant, Goya, Picasso; we all know). Even if we study their early, even very early and childish, works we seem to find no formal mistakes. But perhaps one should not use the term mistake. M. Mendés France (inspired by Erik Dietman) questions whether artists can make mistakes at all, [11], Vol.I.

Many can learn and, e.g. the 19th century was full of gifted and informed 'amateurs' who complemented their activity (e.g. traveling) and profession by sketching.

[Here I included several examples: H. C. Andersen, Ch. Baudelaire, M. Proust, J. W. Goethe, V. Hugo, and showed also some less known (contemporary) 'amateurs'.]

Times are changing, less training and less desire in the age of 'instant picture taking' leaves us perhaps with pictograms (and souvenir photography) and even with graph-drawing as the modern equivalent of some of these old-fashioned skills.

What a chance for Graph Drawing!

2.4 Eternal Style and Quality

It is perhaps surprising to what degree the quality of those sketches didn't change over time. In the same way as Shakespeare and even Greek drama and poetry seem to be forever focused on the actual themata, the sketches show a surprising robustness and unity of style over the ages. Perhaps it is due to the fact that a sketch is often a direct witness of the latent mental power, of the artistic tendency of an artist,

One can only speculate to which degree the prehistoric stone drawings and the Etruscan heads belong to the category of sketches. But perhaps the extent to which these works are directly bounded to the mental processes and the destiny of individuals influenced their acceptance by modernists early in this century (which then started their popularity).

But sometimes we do not have to speculate and we can document and prove these claims.

To be more specific I want to illustrate this with the sketches of Magister Theodoricus – one of the masters of the international Gothic. Theodoricus was the court painter of Charles IV and he decorated his newly built castle Karlštejn (approx. 35 km west of Prague) during the period 1350-80. The most precious

room in this castle, the Holy Cross Chapel, (constructed under influence of Sainte Chapelle in Paris) is decorated by 129 large format paintings on wood panels. These pictures represent portraits of saints and as a whole represent one of the largest collections of its type. (See [12] for a scholarly description of the work and historical context of Magister Theodoricus.)

Under these panels one finds plastered wall with a few preserved sketches by Theodoricus and his workshop. These sketches probably helped to design the complex setting of the chapel and possibly aided in the right positioning of portraits of saints (how much different from today's CAD praxis).

But the comparison of these sketches – wall drawings and final works is really striking. While the final wooden panels are beautiful examples of art of its time (international Gothic, i.e. icon - type painting, rigid, with golden highly decorated background, and with strikingly realistic faces) the sketches are surprisingly modern and some seem to share the style and qualities of a Renaissance drawings.

This is not a singularity and it can be observed elsewhere. Many old sketches look strikingly 'modern'. On the other hand many modern sketches resemble old times. It is as if the times were not changing (for sketches).

2.5 Rarity and Inaccessibility

Sketches are difficult to see. Artists consider sketches their private diaries and consequently do not want to part with them, they are guarded authors secrets. Thus in turn they are valuable collectibles and only few have reached facsimiled, not to mention accessible, editions.

[Here I gave examples of Delacroix, Tichý, Cézanne and Gauss sketchbooks.]

Public collections do not usually show sketches because of their fragility. So we have to rely on reproductions and rare (usually thematic) exhibitions. And this goes hand in hand with the effectiveness of sketching: just a few lines (of Matisse or Picasso, of course, but count how many lines are used in some of the beautiful realistic drawings of Rembrandt) which make a distinction!

A good drawing or a good sketch is a jewel. So should be its technological counterpart. This calls for a hierarchical *classifications* of drawing devices and programs. Art is not an area very remote from Algorithm Design. This fits to our *Creative Thesis* (see [8], [9]; for supporting evidence see e.g. [7]). But this gnoseological reason has to be deepened and complemented so that it will permit an action (transformed in modern terms to algorithms).

3 Summary (Of the First Part of the Lecture)

Sketches, in their technological simplicity and predominantly line character, stand perhaps closest to graph drawing. Linear drawing is the main domain of sketches (despite of the versatility and broadness of this artistic category).

The Art of Sketches has been developing for centuries and their Style shows a surprising tranquility. The discoveries in this area seem to have stellar qualities

and seem to have the character of fundamental discoveries. Probably this is caused by the fact that sketches are so closely related to our perception and mental processes.

I believe this calls for new investigations of these techniques and new approaches to classical branches of art history so that these achievements can be applied to modern technology, computer graphics and, last but not least, graph drawing.

And also recall that people mostly enjoy and value sketches and they believe that they understand them (as opposed to the large part of modern technology).

I believe this calls for

Analysis of the Technology of Artistic Perception

Analysis of Sketches (more generally of Artistic Works) from the point of view of Visual (Technological) Function

Computer generated art addresses some of these phaenomena but these questions call for thorough and scientifically exact investigations guided by the hope that it will be possible to apply the quantified results to technology (Artificial Intelligence and, also, Graph Drawing).

Instead of postmodern archaeology of modern art ([3]) we need *technology of art* and of artistic processes in general.

In the same vein Artificial Intelligence should capture *technology of knowledge* as opposed to archaeology of knowledge (Foucault, [4]).

It is my feeling that all modernism gets its fulfillment in Artificial Intelligence. AI should inspire *technology* of intelligence, technology of knowledge. Kosuth ([7] goes in the right direction and is interesting (but as was realized early unfortunately it is full of loopholes).

Algorithms for graphical outputs should take into account quantifications and the underlying structural essence of highly successful drawing activities – sketching, drawing techniques (and, yes, their visual and mental tricks) developed during the long history of their development.

And even if such quantification would appear impossible to achieve (by present means) it should provide some criteria for perception of technological principles of art and allow to find algorithms for satisfactory drawings. The goal may be not to lay down rules but to show what to look for and what to achieve. I started to pursue these questions at both ends of the spectrum – artistic as well as algorithmic.

Due to the flexibility and qualities of graphic outputs of computers perhaps in the future art can influence technology directly – perhaps for the first time in history. Since Marcel Duchamp we are already prepared by *freedom of indifference*, [2]. We have to approach Graph Drawing with the indifference reserved for Readymades. But the analysis has to be deeper, the indifference greater as we want to construct, to produce, to repeat. Thus technology, i.e. *technology of art*.

We need art (theory) without aesthetic, which will perhaps lead to aesthetic without art (i.e. technology of artistic perception).

Of course matters are not simple (as they never were) but we have to try.

[For elaboration of these rather sketchy lines see [10].]

[There are many more applications of Science to Art. These are presently often misunderstood, even misused (e.g. recently fractals or complexity) but they are more frequent. This was elaborated in the second part of my lecture, see [10] and [11].]

4 Appendix

On the occasion of two 1998 Prague exhibitions and a two - part exhibition in Rakovník (December 1998-March 1999) a two volume catalogue *Anthropogeometry I, II* was published by Rabasova Gallery, [11]. The Volume I contains texts related to the work of authors (contributions by prominent Czech art-historians I. Neuman, E. Petrová, B. Jiráčková, M. Nešlehová), and texts related to the problems of art and science in general (contributions by H. Damisch, M. Mendès France and J. Nešetřil). The Volume II contains reproductions of some of our works divided in four parts (I. Large Formats, II. Stories, III. Geometry, IV. Sketches and Miniatures) approx. 90 reproductions in total.

The volumes also contain the following text which became known as a 'manifesto'.

Nešetřil and Načeradský 1995 Manifesto

The relationships between scientific fields and art trends are becoming increasingly topical, possibly as a result of general uncertainty, a lack of aims and the fragmented nature of people's experiences. This is perhaps (paradoxically) why many artists try to arrange and formulate the principles of their work in a more precise manner, for instance by logical means. On the other hand, scientists and mathematicians in particular like to emphasize the intuitive and aesthetic features of their work, something that can be seen in many meetings, partial formulations, seminars and texts. Our current work is different. We confirmed the usefulness of our contacts during our earlier collaboration (Ateliér 1993, Galerie Artforum 1992, Galerie Na Bidýlku 1995). What we are now concerned with is the attempt at a new form of depiction, a new way of seeing, a different pictorial construction that will enable the capturing the world of doubts, one of fragmentary experiences and confusion of visual information. We are looking for construction and method. We feel an affinity towards Poussin's struggle, as well as towards the complex designs of Raphael and Rubens which encourage us and confirm that we are attempting the possible. We feel a great affinity towards the struggle of the Cubists and Futurists, who are our 'fathers' although we are going elsewhere. We found partial confirmation of our techniques and methods in the lines of Braque that had previously remained a mystery. This was very important! We follow contemporary geometry in all its fields. Encouragement is to be found even here! Its bold constructions give us courage. We try to return to painting its fundamental and eternal problem - the problem of depiction and above all the depiction of space. We don't sense a crisis, only a limitation of our own energy and ability. We are not theoretists, but workers on a construction site. There is nothing more to reveal at this point.

(First published by Galerie VIA Art, 1995, translated by R. Drury.)

References

[1] Hubert Damisch: The origins of Perspective, MIT Press, 1994, original French edition Flamarion 1887
[2] Marcel Duchamp: The writings of Marcel Duchamp (ed. M. Sanouilet, E. Peterson), Oxford University Press 1973
[3] Thierry de Duve: Kant after Duchamp, MIT Press 1998
[4] Michel Foucault: The Archeology of Knowledge, Pantheon Books, New York 1972
[5] Michel Mendès France, Jaroslav Nešetřil: Fragments of a Dialogue, KAM Series 95–308
[6] Wassily Kandinsky: Punkt und Linie zur Fläche, Bauhaus Books, reprinetd (in English) Dover 1979
[7] Joseph Kosuth: Art after Philosophy I,II. In: Idea Art (ed. G. Battcock), Dutton, 1973
[8] Jaroslav Nešetřil: Mathematics and Art, Universität Bonn, 1991
[9] Jaroslav Nešetřil: An Essay on Art and Mathematics, From the Logical Point of View, 2, 2/93 (1994), 50 – 72
[10] Jaroslav Nešetřil: Art of Drawing, KAM–DIMATIA Series 99-437
[11] Jaroslav Nešetřil, Jiří Načeradský: Anthropogeometry (2 volumes), Rabasova Gallery, Rakovník–Praha, 1999
[12] Magister Theodoricus – Court Painter to Emperor Charles IV (ed. J. Fajt), Národní Galerie, Praha, 1997

An Heuristic for Graph Symmetry Detection

Hubert de Fraysseix

CNRS UMR 8557
E.H.E.S.S.
54 Bd Raspail
75006 Paris
France
hf@ehess.fr

Abstract. We give a short introduction to an heuristic to find automorphisms in a graph such as axial, central or rotational symmetries. Using techniques of factorial analysis, we embed the graph in an Euclidean space and try to detect and interpret the geometric symmetries of of the embedded graph. It has been particularly developed to detect axial symmetries.

1 Introduction

Testing whether a graph has any axial (rotational, central, respectively) symmetry is a NP-complete problem [15]. Some restrictions (central symmetry with exactly one fixed vertex and no fixed edge) are polynomialy equivalent to the graph isomorphism test. Notice that this latter problem is not known to be either polynomial or NP-complete in general. For isomorphisms, several heuristics are known (e.g. [4]) and several restrictions leads to efficient algorithms: linear time isomorphism test for planar graphs [11] and interval graphs [14], polynomial time isomorphism test for fixed genus [18, 7], k-contractible graphs [20] and pairwise k-separable graphs [19], linear axial symmetry detection for planar graphs with a given embedding. Symmetry detection and display for trees and, more generally, for outerplanar graphs, serie-parallel digraphs and planar graphs may be found in [16][17][9][10].

In the context of automatic generation of industrial diagrams, the aim of symmetry display is not only to get a 'nice' drawing but also to highlight some of the semantic of the design. Usually the given networks are not planar and even so, due to other constraints (even aesthetics ones), crossings are very common.

We present here the main outlines of an heuristic to detect symmetries. A mathematical justification is to appear in [5], where we propose an exact algorithm. A specific version will soon be integrated in PICTEL, our diagram drawing module of CATIA®[1] but has already been integrated in our research graph drawing toolkit PIGALE[2].

[1] CATIA is a registered trade mark of IBM

[2] PIGALE is a Public Implementation of a Graph Algorithm Library and Editor which is available by anonymous FTP at `ftp://pr.cams.ehess.fr/pub/pigale.tgz`

J. Kratochvíl (Ed.): GD'99, LNCS 1731, pp. 276–285, 1999.

Our main idea to exhibit the symmetries of a graph (planar or not) is to define a distance among the vertices of the graph and embed it isometrically in $\mathbb{R}^{n-1}$, where n is the order of the graph. Then detecting the symmetries of the graph mainly reduces to finding the geometric symmetries of the embedded graph and checking that they do corresponds to symmetries of the given graph.

2 Automorphisms and Symmetries

An *automorphism* of $G = (V, E)$ is a permutation σ of V and E, such that:

- $\sigma(E) = E$,
- $\sigma(V) = V$,
- $e \in E$ is incident to $v \in V \iff \sigma(e)$ is incident to $\sigma(v)$

The usual meaning given to axial, central and rotational symmetry is strongly related to a drawing of a graph. We shall say that such symmetries exist if and only if there exists a drawing of the graph in the plane (where all vertices are represented by different points and edges by lines which may cross) in such a way that a usual geometric axial, central or rotational symmetry appears. A *central symmetry* is an involutive automorphism $\sigma \in \mathrm{Aut}(G)$, such that the fixed points set of σ is either included in E or reduced to a single vertex.

A *rotational symmetry* is an automorphism $\sigma \in \mathrm{Aut}(G)$, which cycles are all of the same length (greater than two) and such that the fixed points set of σ is either empty or reduce to a single vertex. The length of the cycles is the *order* of the symmetry and the rotational symmetry will be said to be *even* or *odd* depending on the evenness of its order.

An *axial symmetry* is an involutive automorphism $\sigma \in \mathrm{Aut}(G)$, such that the subgraph of G induced by the fixed points set of σ is embeddable on a line.

A *planar symmetry* is a symmetry which appears as a geometric symmetry for some planar embedding of the graph.

Remark 1. An even rotational symmetry define axial symmetries but an odd one may define none.

3 The Embedding Model

We shall recall some very basic methods used in *Factorial Analysis* applied to the study of statistical data [3][12].

3.1 Euclidean Semi-Distances

Let X be a set of n points. A *semi-distance* d on X is a mapping which satisfies all the axioms of a distance but one: two distinct points are allowed to be at null semi-distance. The semi-distance d is *Euclidean*, if there exists an isometry from X to $\mathbb{R}^{n-1}$: the restriction of the usual distance of $\mathbb{R}^{n-1}$ to the image of X coincides with the given semi-distance d on X.

Given a semi-distance d on a set X, it is classical to define the corresponding inner-product matrix W define by the formula:

$$W_{i,j} = \frac{1}{2}\big(d^2(v_i,\cdot\,) + d^2(v_j,\cdot\,) - d^2(\cdot\,,\cdot\,)\big) \tag{1}$$

where

$$d^2(v_i,\cdot\,) = \frac{1}{n}\sum_{k=1}^{n} d^2(v_i, v_k)$$

$$d^2(\cdot\,,\cdot\,) = \frac{1}{n}\sum_{i=1}^{n} d^2(v_i,\cdot\,)$$

It is well known that the semi-distance d is Euclidean if and only if the matrix W defines a positive semi-definite bilinear form [1][8][13], that is all its eigenvalues are positive or null. The rank of W (i.e. the maximum of non null eigenvalues) is bounded by $n-1$, the maximal dimension of the vector space generated by n points.

If we denote by $F_1, \ldots, F_p$ an orthonormal basis of eigenvectors associated to the strictly positive eigenvalues $\lambda_1, \ldots, \lambda_p$, we have:

$$d^2(v_i, v_j) = \sum_{k=1}^{p} \lambda_k (F_{i,k} - F_{j,k})^2 \tag{2}$$

3.2 Defining a Semi-Distance in a Graph

Seeking a *natural* semi-distance between vertices of a graph, one could consider the length of the shortest paths joining them. But even the complete graph on 4 vertices with one edge deleted proves that shortest paths do not define an Euclidean semi-distance.

Many mathematical statisticians have define Euclidean semi-distances on abstract sets. But from previous work, while supervising P. Kunz's thesis [13] on the problem of partitioning large graphs into a given number of subgraphs [6], the so called Czekanovski-Dice semi-distance seemed to be very appropriate to reveal the structure of a graph. With respect to that semi-distance, two vertices are close to each other if they have many common neighbors: For each pair (v_i, v_j) of vertices, we denote N_i the set of the neighbors of v_i:

$$N_i = \{v_k \in V, \quad (v_i, v_j) \in E\} \cup \{v_i\}$$

Then, the semi-distance d is defined by:

$$d^2(v_i, v_j) = 1 - 2 \cdot \frac{|N_i \cap N_j|}{|N_i| + |N_j|} \tag{3}$$

Remark that two adjacent vertices having the same neighbors are at semi-distance 0 and that two non adjacent vertices having no common neighbor are at semi-distance 1. Notice that the semi-distances are obviously preserved by any automorphism of the graph.

3.3 Embedding and Projecting the Graph on Planes

The first step of the heuristic is to compute the Czekanovski-Dice semi-distance among the vertices of the graph and the corresponding inner-product matrix W. As the Czekanovski-Dice semi-distance is Euclidean, we can diagonalise W whose eigenvalues are all positive or null. Unless G is a complete graph, the rank of W is not null, and if G has more than 2 vertices its rank is at least 2, but if there are many pairs of vertices with the same neighbors, the rank of W decreases and we get many null eigenvalues.

We then compute an orthonormal basis and compute the p coordinates of all the vertices corresponding to the p strictly positive eigenvalues. We then sort the axis in decreasing order according to the value of their corresponding eigenvalue. Using the terminology of mechanics, the origin is the inertia center, the axis are the principal axis of the set of points and the inertia along each axis decreases with respect to that order. The main tool of Factorial Analysis is precisely to study the projections of the points on the planes defined by the first axis which reflects the main semi-distances structure.

So the next step of the heuristic then consists in looking at the geometric symmetries that may appear when projecting the points on the planes defined by the first axis.

4 The Heuristic by Examples

Paradoxically, it is far easier to find a symmetry in a graph with a small automorphism group (when one exists!). If it is not the case, there are many geometric symmetries in the embedded graph and many equal eigenvalues (e.g the Petersen graphs has, among its nine eigenvalues, only two distinct ones). If an eigenvalue has a multiplicity greater than one, there is no canonical way to compute the corresponding eigenvectors and heuristics have to be designed to select the most appropriate ones such that in some plane defined by them a geometrical symmetry appears.

4.1 All the Eigenvalues are Distinct

This is the easiest case, as all the principal axis are uniquely determined. In all the cases we encountered, it is only necessary to check for a geometric axial symmetry of the graph projected on the first two principal axis (x, y), or to check for the identity. Depending on the number of edges between the two isomorphic graphs defined by the axial symmetry, the geometric symmetry appears on the y-axis or the x-axis. The points with null coordinates are the fixed points, and the other points split into two sets (the one with positive x-coordinates and the ones with x-negative coordinates). Usually the geometric symmetry is defined by the y axis: the fixed points are the points with null x-coordinate, and the two isomorphic graphs are defined by the points of positive (resp. negative x-coordinates). If no two points have the same x-coordinate, the identification of

the corresponding symmetric vertices is trivial. Otherwise the third axis usually breaks the ambiguities. In a few cases, the edges joining the fixed points, are not a chain subgraph (i.e. the axial symmetry detection fails) but we do hope to solve this problem by doing a closer analysis of the fixed points set. We never found an example where geometric symmetry did not correspond to an automorphism of the graph. Figure 1 is a typical example where the first eigenvalues are distinct. The image on the left side represents the projection on the first two principal axis, and on the right side we show the graph where the vertices have been relabelled according to the symmetry. The drawings of the planar graphs use the so-called Tutte-Circle algorithm of PIGALE (the postscript files were created by the LEDA library).

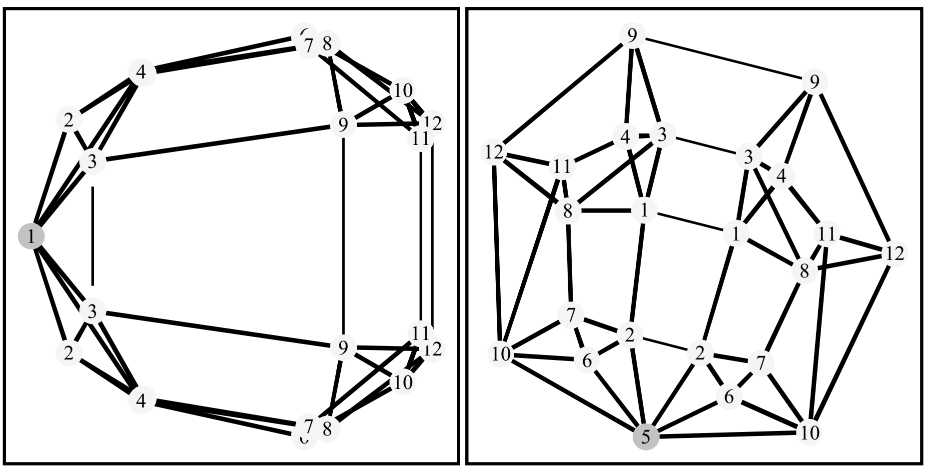


Fig. 1. A non-planar graph with only an axial symmetry

4.2 The First Eigenvalue has Multiplicity Two

In such a case, we highly suspect the existence of a central symmetry or a rotational symmetry. For a central symmetry, we only have to check that there exists at most one point with x and y null coordinates and that the points have a geometric central symmetry.

Then we try to exhibit a rotational symmetry. This is also very simple: we consider the points which are at maximal distance of the origin and sort them according to their polar angle and we try to match them by rotations (matching-pattern type algorithm). If we succeed, we check that the matching extends to all the points.

If we find a central or rotational symmetry, we try to exhibit an axial symmetry. As the two first eigenvalues are equal, the position of the first axis depend on the labelling of the vertices and has nothing to do with its structure. So we

really need an heuristic to find out a rotation of the first two axis such that a geometrical axial symmetry appears along one of those axis in the plane they define. If we do have an even rotational symmetry, the problem is extremely simple. Otherwise it seems rather difficult and we are far from having an algorithm that we could guaranty. Nevertheless, we do have an heuristic that has been efficient in all the cases we met. It is probably due to the fact that we only consider rotational symmetries when the first two eigenvalues are equal.

Figure 2, Figure 3, Figure 4 gives examples where the first eigenvalue is multiple. The Factorial Analysis drawings (left of the pictures) are done after the correct rotation of the axis has been applied. (We only show one or two symmetries found by the heuristic).

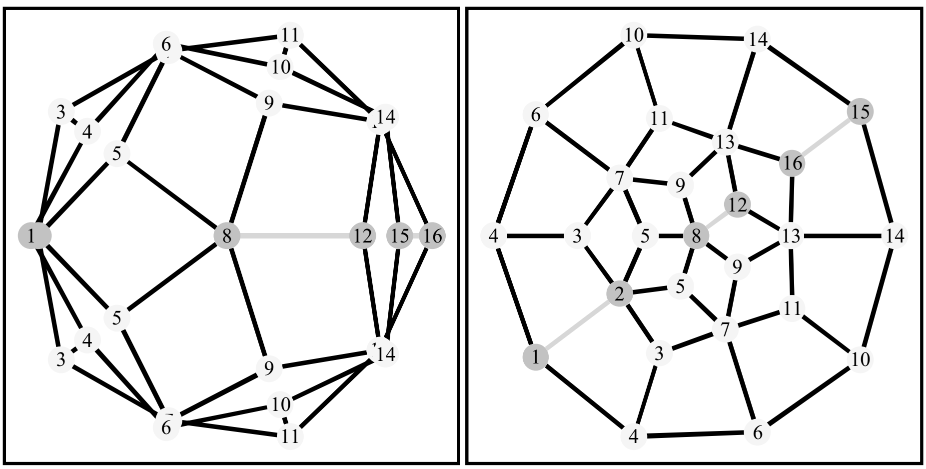


Fig. 2. A planar graph with rotational and axial symmetries

4.3 General Case

If the first eigenvalue has multiplicity one, and the second one has multiplicity two, we are more or less in the previous situation.

If all the eigenvalues of a graph are equal, we get no information from any projection, but we suspect that the graph is vertex transitive (i.e. for any pair of vertices there is an automorphism matching them).

As we said earlier, if two adjacent vertices have the same neighbors, they have the same coordinates and so the null eigenvalue has a multiplicity greater than one. If the null eigenvalue has a big multiplicity, we probably could find interesting automorphisms.

Otherwise, we do have examples where our heuristic completely fails (e.g. Petersen graph, complete bipartite graphs etc).

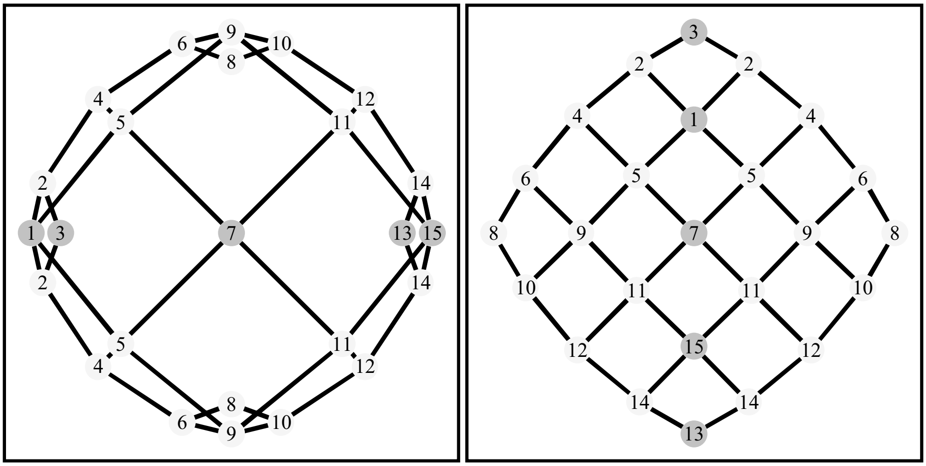


Fig. 3. A grid graph with one of its axial symmetries

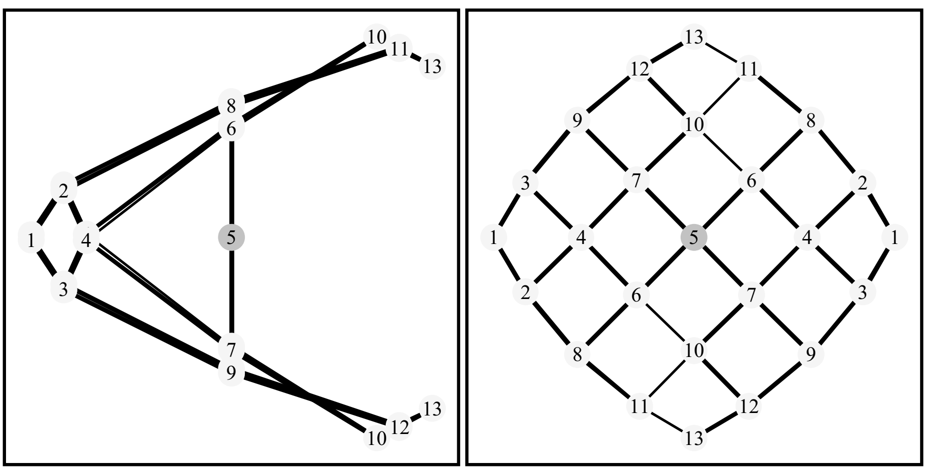


Fig. 4. A grid graph with a central symmetry

4.4 More Pictures

The following pictures, represent famous graphs taken from *Examples and Counterexamples in Graph Theory* [2]:

- Figure 5 represents the Folkman graph.
- Figure 6 On the left, a graph whose automorphism group is the Klein 4-group, and a graph whose automorphism group is C_6.

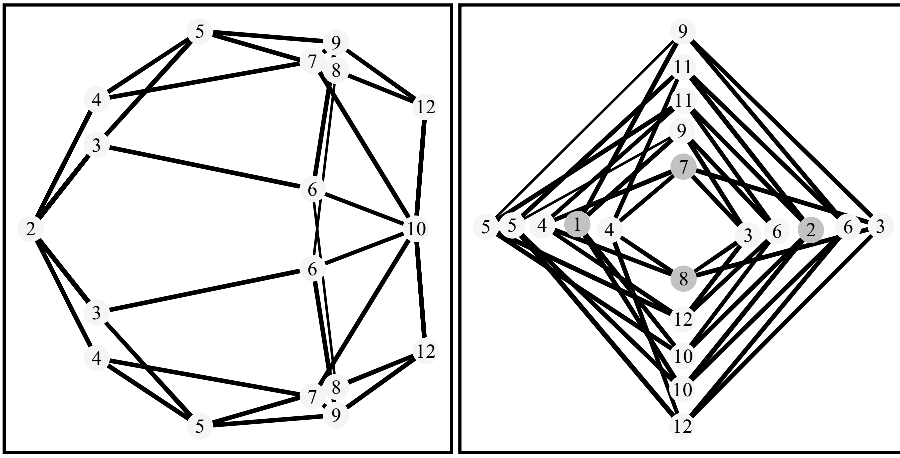


Fig. 5. The Folkman graph and one its Factorial Analysis projection

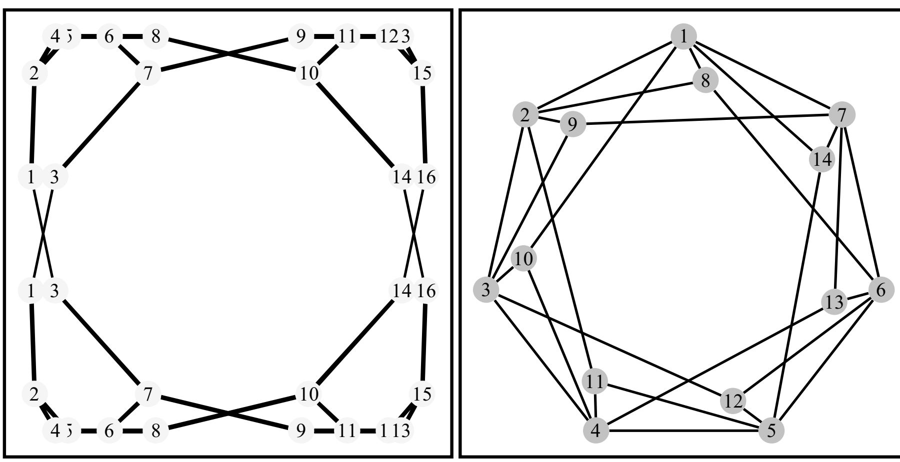


Fig. 6. Two other Factorial Analysis projections from [2]

4.5 Open Questions

- Can two non-isomorphic graphs have the same semi-distances among their vertices?
- Is there a graph with a rotational or central symmetry that does not have at least two equal eigenvalues?
- Is there a graph having all its eigenvalues equal which is not vertex transitive (i.e. for any pair of vertices there is an automorphism matching them)?
- If a graph have two adjacent vertices with the same neighbors, the null eigenvalue has a multiplicity greater than one. Characterize the other configurations that imply that the null eigenvalue has a multiplicity greater than one.

4.6 Concluding Remarks

Sometimes, a geometric axial symmetry only reveals an automorphism as the set of fixed points may not be a subgraph of a chain. This happens very often when the graph is a tree, but we shall implement the linear time algorithm to exhibit axial symmetries in such cases. As stated before, the heuristic often fails if the first eigenvalues have a great multiplicity. In such cases a higher complexity algorithm should be devised. Otherwise this heuristic seems very effective and do find symmetries on all the examples we found in the algorithmic literature on that subject and on nearly all the examples we constructed.

We also experimented the heuristic on random maximal planar graphs (using the LEDA generator): As expected, most of such graphs with few vertices do have symmetries and there are very few as soon as the vertex number exceeds twenty. More precisely, the proportion of random maximal planar graphs that do have symmetries is roughly 95% (resp. 70%, 35%, 15% ,5%) when the number of vertices is 8 (resp. 10, 12, 14, 16).

The overall complexity of the heuristic is $\mathcal{O}(n^3)$, (i.e. the complexity of the computation of the eigenvalues). It certainly could be improved, but that was not our main goal. The different heuristics to identify the isomorphic subgraphs, compute the rotations etc are either linear or in $\mathcal{O}(n \log n)$. To give a rough idea of the time computation, on a Pentium running at 200MHz, finding a symmetry in a graph with 100 vertices takes 0.45 seconds and in a graph with 400 vertices 32 seconds. The code source of the C^{++} implementation is less than 1500 lines.

Acknowledgment

The author would like to express his gratitude to J.P. Benzécri who introduced him the Factorial Analysis techniques, to Jarik Nešetril and to P. Ossona de Mendez for his helpful comments and remarks.

References

[1] J.P. Benzécri and et al, *L'analyse des données — Tome II: l'analyse des correspondances*, Dunod, Paris, 1973.
[2] M. Capobianco, *Examples and counterexamples in graph theory*, Elsevier, North Holland, New-York, 1978.
[3] C.N.R.S. (ed.), *L'analyse factorielle et ses applications*, Paris, 1956.
[4] D.G. Corneil and C.C. Gotlieb, *An efficient algorithm for graph isomorphism*, Journal of the ACM **17** (1970), no. 1, 51–64.
[5] H. de Fraysseix and P. Ossona de Mendez, *Discovering automorphisms using abstract distances*, Tech. report, KAM-DIMATIA Series, 1999, (in preparation).
[6] H. de Fraysseix and Kuntz P., *Pagination of large scale networks*, Algorithms review **2** (1992), no. 3, 105–112.
[7] I.S. Filotti and J.N. Mayer, *A polynomial-time algorithm for determining the isomorphism of graphs of fixed genus (working paper)*, Conference Proceedings of the Twelfth Annual ACM Symposium on Theory of Computing (Los Angele, California), 1980, pp. 236–243.

[8] S.L. Hakimi and S.S. Yau, *Distance matrix of a graph and its realizability*, Quaterly of Applied Mathematics **22** (1964), 305–317.

[9] S-H. Hong, P. Eades, and S-H. Lee, *F*inding planar geometric automorphisms in planar graphs, ISAAC 98 (ed. Chwa and Ibarra), Springer Lecture Notes in Computer Science 1533 (1998), 277–286.

[10] S-H. Hong, P. Eades, and S-H. Lee, *D*rawing series parallel digraphs symmetrically (to appear).

[11] J.E. Hopcroft and J.K. Wong, *Linear time algorithm for isomorphism of planar graphs (preliminary report)*, Conference Record of Sixth Annual ACM Symposium on Theory of Computing (Seattle, Washington), 1974, pp. 172–184.

[12] J.B. Kruskal and J.B. Seery, *Designing network diagrams*, Proceedings of the First General Conference on Social Graphics, 1978, pp. 22–50.

[13] P. Kuntz, *Représentation euclidienne d'un graphe en vue de sa segmentation*, Ph.D. thesis, Ecole des Hautes Etudes en Sciences Sociales, Paris, 1992.

[14] G.S. Lueker and S.B. Kellogg, *A linear time algorithm for deciding interval graph isomorphism*, Journal of the ACM **26** (1979), no. 2, 183–195.

[15] J. Manning, *Geometric symmetry in graphs*, Ph.D. thesis, Purdue University, New York, 1990.

[16] J. Manning and M.J. Atallah, *Fast detection and display of symmetry in trees*, Congressus Numerantium **64** (1988), 159–169.

[17] ———, *Fast detection and display of symmetry in outerplanar graphs*, Discrete Applied Mathematics **39** (1992), 13–35.

[18] G.L. Miller, *Isomorphism testing for graphs of bounded genus*, Conference Proceedings of the Twelfth Annual ACM Symposium on Theory of Computing (Los Angeles, California), 1980, pp. 225–235.

[19] ———, *Isomorphism of graphs which are pairwise k-separable*, Information and Control **56** (1983), no. 1/2, 21–33.

[20] ———, *Isomorphism of k-contractible graphs. a generalization of bounded valence and bounded genus*, Information and Control **56** (1983), no. 1/2, 1–20.

Isomorphic Subgraphs

Sabine Bachl

University of Passau, 94030 Passau, Germany
Sabine.Bachl@informatik.uni-passau.de

Abstract. We are interested in finding symmetries in graphs and then use these symmetries for graph drawing algorithms.
There are two general approaches to this problem, the first one is known as GEOMETRIC SYMMETRIES on the basis of drawings, the other rests upon the graph-theoretical notion of graphs. For a given graph G the ISOMORPHIC SUBGRAPHS problem makes use of the second approach and tries to find the two largest disjoint isomorphic subgraphs in G. Hence, G consists of two identical copies and a remainder.
There are many NP-complete or open problems related to our problem, like GRAPH ISOMORPHISM, GRAPH AUTOMORPHISM or LARGEST COMMON SUBGRAPH.
We show that the ISOMORPHIC SUBGRAPHS problem is NP-hard for connected outerplanar graphs, and 2-connected planar graphs and is solvable in linear time when restricted to trees.
Additionally we will shortly discuss the applicability of ISOMORPHIC SUBGRAPHS in graph drawing algorithms.

1 Introduction

As graphs are used in various application areas, symmetric drawings are desirable for a better and faster understandability ([17],[16]). Therefore, symmetry is one of the major aesthetics in graph drawing apart from edge crossing, edge length and routing. There are two different approaches to symmetry. The first one is based on the drawing itself, the second rests upon the graph-theoretical notion of graphs.

In the first approach, symmetry in drawings is defined as a geometric property, using rotations or reflections. This has been investigated in depth by Manning [13] and by Manning and Atallah ([14],[15]) for trees, outerplanar graphs and embedded graphs. They proved that several variants of GEOMETRIC SYMMETRIES are NP-complete. Lipton et al. [11] described a model for measuring the symmetry of straight-line drawings.

An investigation on symmetry is also possible on the basis of the graph-theoretical definition of a graph $G = (V, E)$ as a set of nodes and a set of edges. This view leads to the notion of isomorphism, automorphism or homomorphism. Throughout the last decades many people have contributed to this area.

First of all there is the GRAPH ISOMORPHISM problem, which is one of the challenging open problems in complexity theory ([8],[20]). GRAPH ISOMORPHISM

J. Kratochvíl (Ed.): GD'99, LNCS 1731, pp. 286–296, 1999.

is neither known to be NP-complete nor exists a polynomial algorithm to solve this problem. A detailed summary of various heuristics can be found in [18]. They are all based on so-called graph invariants which remain unchanged under isomorphism, like node degree etc.

A well-known NP-complete problem in this field is SUBGRAPH ISOMORPHISM (GT48 in [6]), containing CLIQUE, COMPLETE BIPARTITE SUBGRAPH or HAMILTONIAN as a special case. SUBGRAPH ISOMORPHISM is also NP-complete for outerplanar graphs ([21]), sets of chains and forests ([6]). There exist polynomial algorithms if both graphs are trees ([19]) or 2-connected outerplanar ([10]).

Another related NP-complete problem is LARGEST COMMON SUBGRAPH (GT49 in [6]). It is only polynomial solvable if the graphs are trees. Levi and Luccio ([9]) proposed a heuristic which is also based on graph invariants.

Further related NP-complete problems are ISOMORPHISM WITH RESTRICTIONS ([12]), MAXIMUM SUBGRAPH MATCHING (GT50 in [6]), GRAPH-K-ISOMORPHISM ([24]), GRAPH AUTOMORPHISM and many variants ([12]).

The ISOMORPHIC SUBGRAPHS problem can be described as searching for the two largest isomorphic disjoint subgraphs in a given graph. It is defined formally in Section 2. There exist two variants of the problem: IES and INS where the subgraphs can be either edge-induced or node-induced (see Section 2).

Most of the known NP-complete problems in this area except GRAPH AUTOMORPHISM and GEOMETRIC SYMMETRIES rest upon two given graphs. In the ISOMORPHIC SUBGRAPHS problem only one graph is given and therefore the cut-line must be found in order to reduce our problem to the known NP-complete problem. One might think, that the ISOMORPHIC SUBGRAPHS problem is a variation of the well-known SUBGRAPH ISOMORPHISM problem with graphs G and H now taking the disjoint union $G \cup H$. However, this not true in general. $G \cup H$ may contain isomorphic subgraphs which are each larger than G. Easy reductions to the other similar NP-complete problems fail, too.

For the purpose of displaying symmetries in graphs, GRAPH AUTOMORPHISMS are too restrictive. A few nodes or edges may destroy a nontrivial automorphism. The ISOMORPHIC SUBGRAPHS problem is a less restrictive approach where it is possible to have a remainder in the given graph consisting of unused nodes and edges.

At first sight, GEOMETRIC SYMMETRIES are very similar to the ISOMORPHIC SUBGRAPHS problem. But they have only edges in the remainder and additionally these edges must fulfil a certain symmetry along the axis of symmetry. Our remainder consists of nodes and edges and is completely disregarded.

Finding the isomorphic subgraphs is not the only interesting point of view. Its integration in a layout algorithm will be our focus in the future.

In Section 2 we introduce basic notations and definitions. In Section 3 we prove the NP-completeness of the ISOMORPHIC SUBGRAPHS problem for outerplanar connected and planar 2-connected. This improves and complements our proof for general graphs ([3]) which uses dense graphs with $\Omega(n^2)$ edges. The focus of Section 4 is a linear time algorithm for trees. We conclude with open problems and an outlook.

2 Basic Notions

Let $\mathcal{A}$ be an alphabet and let $\$ \notin \mathcal{A}$ be a new symbol.

For a *string* $w = a_1..a_n$ over $\mathcal{A}$ let $|w| = n$ denote its *length*. The *substring* $a_i..a_j$ of w is denoted by $w[i, j]$. If $j = n$, $w[i, n]$ is called *suffix* of w beginning with the i-th symbol.

The *suffix tree* of a string $w\$$ is a rooted tree with $|w| + 1$ leaves, such that every internal node except the root has at least two sons. Every edge is labeled with a nonempty substring of w, such that the labels of edges leaving a node begin with different symbols, and every suffix of $w\$$ is obtained by a concatenation of the labels of the edges on a path from the root to a leaf. The leaf representing the suffix $w[i, n]$ is labeled by i.

Thus, a suffix tree is a compact data structure for the set of suffixes. They are commonly used for an efficient computation of common substrings. We use them for the computation of the largest isomorphic subtrees.

There are several linear time algorithms for the computation of a suffix tree, see e.g. [22], [23] or [7].

Let us now consider an example. The suffix tree of the Fibonacci word $w = abaababa\$$ is shown in figure 1.

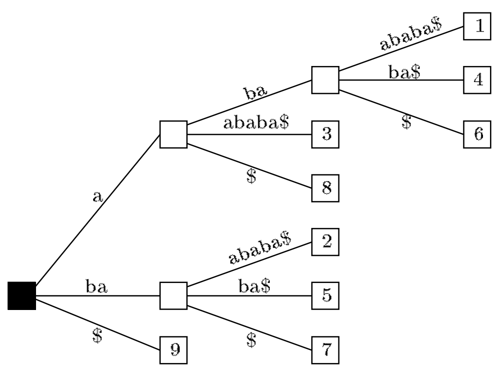


Fig. 1. Suffix tree for the Fibonacci word $w = abaababa\$$.

Next we recall some graph theoretic notions. A graph $G = (V, E)$ consists of a set of nodes V and undirected edges $E = \{\{v_i, v_j\}|v_i, v_j \in V\}$. For convenience, we exclude self-loops and multiple edges. Let the numbers of nodes and edges be denoted by $|V|$ and $|E|$. Two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are isomorphic, if there is a bijection $f : V_1 \to V_2$ such that $\{u, v\} \in E_1$ if and only if $\{f(u), f(v)\} \in E_2$.

Let $G = (V, E)$ be a graph and $V' \subseteq V$. The subgraph $H = G|_{V'}$ consisting of all nodes V' and the edges $E' = \{\{v_i, v_j\} \in E|v_i, v_j \in V'\}$ between the nodes of V' in G is called *Node-Induced Subgraph*.

Let $G = (V, E)$ be a graph and $E' \subseteq E$. The subgraph $H = G|_{E'}$ consisting of all edges E' and their adjacent nodes is called *Edge-Induced Subgraph*.

For our NP-completeness results we reduce 3-PARTITION ([6]):

Instance: A finite set $A = a_1, \ldots, a_{3m}$ of $3m$ elements ($m \in \mathbb{N}$), a bound $B \in \mathbb{N}$, a 'size' $s(a) \in \mathbb{N}$ for each $a \in A$, such that each $s(a)$ satisfies $\frac{B}{4} < s(a) < \frac{B}{2}$ and $\sum_{a \in A} s(a) = m \cdot B$.

Question: Can A be partitioned into m disjoint sets $A_1, \ldots, A_m$ such that, for $1 \leq i \leq m$, $\sum_{a \in A_i} s(a) = B$. (Notice that the above constraints on the item sizes imply that every such A_i must contain exactly three elements.)

We now come to the central notion of this paper: the Isomorphic Subgraphs problems IES and INS.

ISOMORPHIC EDGE-INDUCED SUBGRAPHS, IES

Instance: A graph $G = (V, E)$ and a positive integer K.

Question: Does G contain two disjoint isomorphic edge-induced subgraphs with at least K edges, i.e. are there two sets of edges E_1 and E_2 with $E_1 \cap E_2 = \emptyset$ such that the subgraphs $H_1 = G|_{E_1} = (V_1, E_1)$ and $H_2 = G|_{E_2} = (V_2, E_2)$ are isomorphic, $V_1 \cap V_2 = \emptyset$ and $|E_1| = |E_2| \geq K$.

ISOMORPHIC NODE-INDUCED SUBGRAPHS, INS

Instance: A graph $G = (V, E)$ and a positive integer K.

Question: Does G contain two disjoint isomorphic node-induced subgraphs with at least K edges, i.e. are there two sets of nodes V_1 and V_2 with $V_1 \cap V_2 = \emptyset$ such that the induced subgraphs $H_1 = G|_{V_1} = (V_1, E_1)$ and $H_2 = G|_{V_2} = (V_2, E_2)$ are isomorphic and $|E_1| = |E_2| \geq K$.

3 NP-Completeness of IES and INS

In this section, we prove that IES and INS are NP-complete even for connected outerplanar graphs and two-connected planar graphs. It should be noted that IES is also NP-complete for instances where the graphs are dense ([3]).

3.1 Outerplanar Graphs

Theorem 3.1. *IES is NP-complete for connected outerplanar graphs.*

Proof. We reduce 3-Partition to IES.

Let $A = \{a_1, \ldots, a_{3m}\}$ and $B \in \mathbb{N}$ be an instance of 3-Partition.

We construct a graph G as follows (see Figure 2): The left subgraph L consists of $3m$ fans, where the i-th fan is a chain of $s(a_i)$ nodes, all connected to v_1. The right subgraph R has m fans each consisting of a chain of B nodes connected to v_2. There is an edge between v_1 and v_2. Note that the constructed graph is outerplanar connected.

In the following proof, the i-th fan of L is called $s(a_i)$-fan, the fans of R are called B-fans.

Let $K = 2Bm - 3m$.

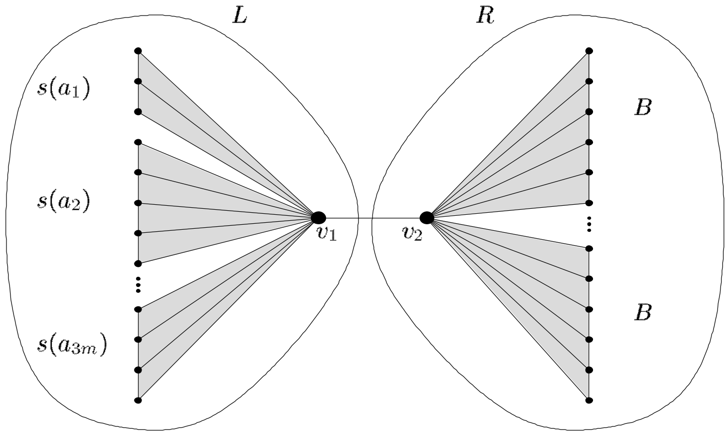


Fig. 2. Reduction to IES for connected outerplanar graphs

Claim. IES has a solution for G and K if and only if A has a 3-Partition.

Proof.

$\Leftarrow$: *Let $A_1, \dots, A_m$ be a 3-Partition of A.*
Define $H_1 = L$ and $H_2 = R$ and a bijection $f : H_1 \to H_2$, such that v_1 is mapped on v_2 and for all $a \in A_j$ $(1 \leq j \leq m)$ the $s(a)$-fans are mapped completely onto the j-th B-fan. Hence, H_1 and H_2 have at least

$$\sum_{a \in A} (s(a) - 1) + s(a) = 2Bm - 3m = K$$

common edges and IES has a solution with $K = 2Bm - 3m$.

$\Rightarrow$: *Let H_1 and H_2 be a solution of IES and $K = 2Bm - 3m$.*

a) *Then we can prove that one of the graphs is equal to L and the other one is equal to R.*
First, $v_1 \in H_1$ and $v_2 \in H_2$ (w.l.o.g.). Assume that v_1 and v_2 belong to the same subgraph, say H_1. By the construction of G, $deg(v_1) = deg(v_2) = Bm + 1$ and $deg(v) \leq 3$ for all $v \in V \backslash \{v_1, v_2\}$.
Therefore mapping v_1 and v_2 on two nodes of H_2 implies a loss of at least $2 \cdot (Bm+1-3)$ edges. Then H_1 and H_2 have each at most $\left\lfloor \frac{|E|-2\cdot(Bm-2)}{2} \right\rfloor$ edges with $|E| = 4Bm - 4m + 1$. Since for all $B > 1$:

$$\left\lfloor \frac{|E| - 2 \cdot (Bm - 2)}{2} \right\rfloor < K - 3$$

we have a contradiction to the number of edges in every subgraph.
Thus, v_1 and v_2 do not belong to the same subgraph and are mapped onto each other.

It remains to show that H_1 has no node of R and H_2 no node of L. Such a mapping would lead to a loss of at least 4 edges which is again a contradiction to $|E_1| = |E_2| > K$.
Hence $H_1 = L$ and $H_2 = R$.

b) *In order to show that 3-Partition has a solution, it is sufficient to show that exactly three $s(a_i)$-fans must be mapped completely onto one B-fan. Since H_1 has K edges, every $s(a_i)$-fan must be completely mapped onto one B-fan, otherwise we loose at least one edge.*
Since $\frac{B}{4} < s(a) < \frac{B}{2}$, exactly three $s(a_i)$-fans must be mapped onto one B-fan. Hence the 3-Partition has a solution which is given uniquely by the mapping of nodes and fans. □

Theorem 3.2. *INS is NP-complete for connected outerplanar graphs.*

Proof. *We reduce 3-Partition to INS. Since the proof is very similar to the previous one, we only show how to construct the graph G (see Figure 3). The left*

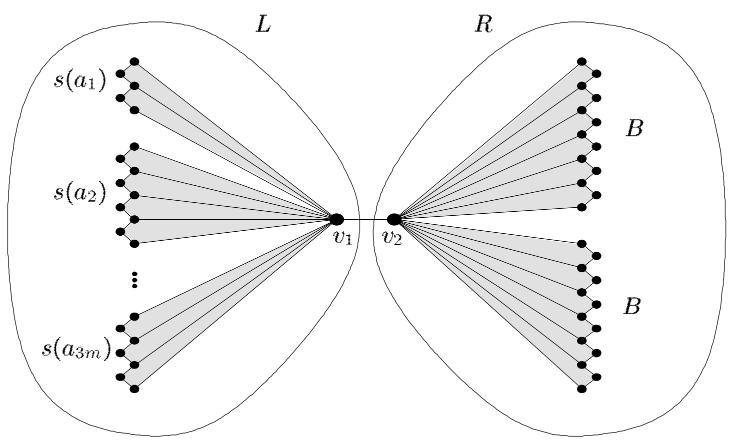


Fig. 3. Reduction to INS for connected outerplanar graphs

subgraph L consists of $3m$ fans, where the i-th fan is a chain of $s(a_i)+(s(a_i)-1)$ nodes and every second node is connected to v_1. The right subgraph R has m fans where each fan has exactly $B+(B-1)$ nodes where every second node is connected to v_2. The nodes v_1 and v_2 are adjacent. The value of K is $3Bm - 6m$.

By the reasoning as above it can be shown that INS has a solution for G and K if and only if A has a 3-Partition. □

3.2 Planar Graphs

IES and INS are NP-complete for connected planar graphs since outerplanar graphs are a subset of planar graphs. Next we show, that they are also NP-complete for 2-connected planar graphs. Again, the proof is similar to the first

one. Therefore, we only present the idea of the proof of IES. The proof of INS is left to the reader.

Theorem 3.3. *IES is NP-complete for 2-connected planar graphs.*

Proof. We reduce 3-Partition to IES.

We construct a graph G (see Figure 4). The left subgraph L consists of $3m$

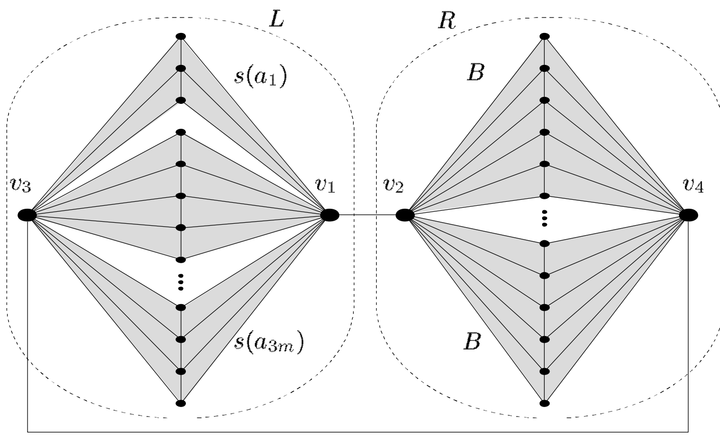


Fig. 4. Reduction to IES for 2-connected planar graphs

fans, where the i-th fan is a chain of $s(a_i)$ *nodes, each connected to* v_1 *and* v_3*. The right subgraph R consists of m fans where each fan has exactly B nodes, each connected to* v_2 *and* v_4*. The nodes* v_1 *and* v_2 *are adjacent as are* v_3 *and* v_4*. Note that G is 2-connected planar. K is chosen as* $3Bm - 3m$*.*

It can be shown that IES has a solution if and only if A has a 3-Partition. In addition to theorem 3.1, we have to prove that v_1 *must be mapped to* v_2 *and* v_3 *to* v_4 *(or* v_1 *to* v_4 *and* v_3 *to* v_2*); otherwise there is a contradiction to the number of edges in the subgraphs.*

□

4 Isomorphic Subtrees

In this section we specialize the Isomorphic Subgraphs problem to trees. We consider free and rooted trees, ordered and unordered trees. For rooted ordered trees the Isomorphic Subgraphs problem is solved by a transformation of trees into strings over pairs of parenthesis and a reduction to the largest identical non-overlapping well-nested substring problem. The other types of trees are reduced to rooted ordered trees.

Let $T = (V, E, r)$ be a tree with root r. A *subtree rooted at* r' $T' = (V', E', r')$ consists of all of the descendants of r' (including r' itself) and their connecting edges.

A part of the following algorithm will be the computation of the longest identical non-overlapping well-nested substrings. The *longest identical non-overlapping well-nested substrings* of a string w are the well-nested substrings $w[i, j]$ and $w[p, q]$ with $w[i, j] = w[p, q]$ and $j < p$ of maximal length. We prove in Lemma 4.1 that they can be computed in linear time on suffix trees. If the context is clear, we will refer to the longest identical non-overlapping well-nested substrings by *identical substrings*.

Lemma 4.1. *Let w be a string over some alphabet $\mathcal{A}$. Then the identical substrings can be computed in linear time.*

Proof. Let T be the suffix tree of w. Let $V^{int} = \{v_1, \dots, v_m\}$ be the set of internal nodes of T.

For every $v_i \in V^{int}$ compute the tuple (v_i, s, P) such that s is the concatenation of the labels from the root to v_i and $P = \{p_1, \dots, p_k\}$ is the set of starting positions of s which is equal to the labels of the leaves of the subtree with root v_i.

Insert the tuple into the list L which is sorted by a descending $|s|$ if $|s| \leq \left\lfloor \frac{|w|}{2} \right\rfloor$. Note that strings with $|s| = 1$ are omitted.

Take the first tuple from L. Determine a possible intersection of the intervals $[p_i, p_i + |s| - 1]$ ($i \in \{1, \dots, k\}$). If there is a p_i and a p_j such that $[p_i, p_i + |s| - 1] \cap [p_j, p_j + |s| - 1] = \emptyset$ and $w[p_i, p_i + |s| - 1]$ and $w[p_j, p_j + |s| - 1]$ are well-nested, the strings $w[p_i, p_i + |s| - 1]$ and $w[p_j, p_j + |s| - 1]$ are the identical substrings. Otherwise take the next tuple out of L and check the conditions again.

As mentioned before the suffix tree can be determined in linear time in the length of the string. The number of internal nodes are at most $|w| - 1$. Therefore the number of tuples is restricted to $O(|w|)$. If $|P| > 2$, $P = \{p_1, \dots, p_k\}$ should be a sorted list such that we only need to compare p_1 and p_k. If these intervals overlap, the innermost intervals do not fit either and the tuple can be omitted. Note that the sorting of L and P can be done in linear time using Bucket Sort. Thus, the computation of the identical substrings of w is in $O(|w|)$. □

Let T be a rooted ordered tree. The isomorphic subtrees T_1 and T_2 of T are computed in three steps. First of all, the tree T is transformed into a well-nested string $s(T) \subseteq \{(,)\}^*$ over pairs of parenthesis using depth first search. Note that every node of the tree is represented by exactly one pair of parenthesis and the isomorphic subtrees are identical well-nested substrings of $s(T)$. After that, the identical substrings $s_1(T)$ and $s_2(T)$ of $s(T)$ are computed with the algorithm given in Lemma 4.1. As every pair of parenthesis represents exactly one node, the isomorphic subtrees can easily be retrieved from the identical substrings by scanning the substrings and its node information.

In the case of free trees, this are trees with no explicitly given root, finding the largest isomorphic subtrees is done by a reduction to the above algorithm for rooted trees. Every node of the tree is chosen once as a root. After that, the maximum of all results is taken. The runtime of this algorithm is $O(n^2)$ then.

In the case of unordered trees, T is first transformed into a right-heavy tree according to the following rules.

If T_1 and T_2 are subtrees rooted at the sons of some node v, then T_1 is left of T_2 if

1. $height(T_1) < height(T_2)$, or
2. $height(T_1) = height(T_2)$ and $size(T_1) < size(T_2)$, or
3. $height(T_1) = height(T_2)$ and $size(T_1) = size(T_2)$ and $level\_weight(T_1) < level\_weight(T_2)$

where *height* is the length of the longest path to some leaf, *size* is the number of nodes and *level_weight* counts the number of descendants by level and returns the least level with different numbers, the number of descendants of T_1 is less than that of T_2.

After that, the above algorithm for ordered trees can be applied. Since the reordering of the tree is linear, the whole algorithm for unordered trees runs in linear time.

5 Open Questions and Outlook

As our next steps we try to find solutions for IES and INS of 2-connected outerplanar and 3-connected planar graphs. We conjecture that these variants are polynomial solvable.

Additionally, we are working on good heuristics for the NP-complete cases of IES and INS in order to achieve some practical results.

Our first approach to a heuristic for IES is adapted from Graph Isomorphism heuristics which are based on so-called graph invariants. These are graph-theoretical properties which remain unchanged under isomorphism.

Since there is normally a remainder in our graph, it is not possible to use these invariants immediately. Instead, we determine the locally best mapping and continue our search with the help of maximal weighted matching algorithms for bipartite graphs.

The rating of a mapping between two nodes v_i and v_j depends for example on the difference of degree, the minimal distance between v_i and v_j and the number of neighbours of v_i and v_j which can be mapped in the next step.

Our idea for the heuristic of INS is similar to the idea of IES except that we operate on faces, not only on nodes. The criteria for good mappings are now for example the number of nodes in the faces or their distance.

Finally, we will try to include the information of isomorphic subgraphs into drawing algorithms. For a better understandability, the isomorphic subgraphs should be drawn identical. For this purpose, we may borrow techniques from the design of hierarchical methods ([1]) and layout graph grammars ([2]). It is also possible to integrate the results into springembedders.

The hierarchical methods are used for drawing one of the isomorphic subgraphs. After that, this subgraph is copied. The integration into the whole graph is done by collapsing the isomorphic subgraphs into two nodes whos sizes are

given by the surrounding rectangles. It is possible now to draw the complete graph and unfold the nodes representing the isomorphic subgraphs afterwards.

Another possibility for drawing algorithms is applicable in springembedders where the position of nodes is determined by forces. Instead of moving only one node in every step, the forces of a mapped pair are determined and both nodes are moved. Thus, the isomorphic subgraphs are drawn identically. The advantage of this method is that the remainder is regarded during the determination of the layout of the isomorphic subgraphs in contrast to the copy-paste-method.

The integration of isomorphic subgraphs into drawing algorithms will also lead to some theoretical problems. One can easily find examples, such that there are no planar drawings of planar graphs, if the isomorphic subgraphs are drawn in the same way. Therefore, we will have to characterize those planar graphs which admit such planar drawings. It might be possible to adapt the results of Feng et al. ([5], [4])

Acknowledgement

I would like to thank Prof. F.J. Brandenburg for his various and detailed comments and suggestions, particularly on the proof of Isomorphic Subgraphs of general graphs.

References

[1] W. Bachl. Entwicklung und Implementierung eines Hierarchischen Springembedders. Diplomarbeit, Universität Passau, 1995.
[2] F. J. Brandenburg. Designing graph drawings by layout graph grammars. In *Proc. Graph Drawing 1994*, Lecture Notes in Computer Science 894, pages 416–427. Springer, 1995.
[3] F.J. Brandenburg and S. Wetzel. On pairs of large common subgraphs. unpublished manuscript, `http://www.fmi.uni-passau.de/~wetzel/paper/`, 1998.
[4] Qing-Wen Feng, Robert F. Cohen, and Peter Eades. How to draw a planar clustered graph. In Ding-Zhu Du and Ming Li, editors, *Computing and Combinatorics COCOON '95*, Lecture Notes in Computer Science 959, pages 21–30. Springer, 1995.
[5] Qing-Wen Feng, Robert F. Cohen, and Peter Eades. Planarity for clustered graphs. In Paul Spirakis, editor, *Algorithms-ESA '95*, Lecture Notes in Computer Science 979, pages 213–226. Springer, 1995.
[6] M.R. Garey and D.S. Johnson. *Computers and Intractability. A Guide to the Theory of NP-Completeness.* W.H. Freeman, 1979.
[7] D. Gusfield. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology.* Cambridge University Press, 1997.
[8] R.E. Ladner. On the structure of polynomial time reducibility. *Journal of the ACM*, 22:155–171, 1975.
[9] Giorgio Levi and Fabrizio Luccio. A technique for graph embedding with constraints on node and arc correspondences. *Information Sciences*, 5:1–24, 1973.
[10] Andrzej Lingas. Subgraph isomorphism for biconnected outerplanar graphs in cubic time. *Theoretical Computer Science*, 63:295–302, 1989.

[11] R. J. Lipton, S. C. North, and J. S. Sandberg. A method for drawing graphs. In *Proc. ACM Symposium on Computational Geometry*, pages 153–160. ACM, 1985.

[12] A. Lubiw. Some NP-complete problems similar to graph isomorphism. *SIAM J. Comput.*, 10(1):11–21, 1981.

[13] J. Manning. *Geometric Symmetry in Graphs*. PhD thesis, Department of Computer Science, Purdue University, 1990.

[14] J. Manning and M. J. Atallah. Fast detection and display of symmetry in trees. *Congressus Numerantium*, 64:159–168, 1988.

[15] J. Manning and M.J. Atallah. Fast detection and display of symmetry in outerplanar graphs. *Discrete Applied Mathematics*, 39:13–35, 1992.

[16] H. Purchase. Which aesthetic has the greatest effect on human understanding. In *Proc. Graph Drawing 1997*, Lecture Notes in Computer Science 1353, pages 248–261. Springer, 1997.

[17] H. C. Purchase, R. F. Cohen, and M. James. Validating graph drawing aesthetics. In *Proc. Graph Drawing 1995*, Lecture Notes in Computer Science 1027, pages 435–446. Springer, 1996.

[18] Ronald C. Read and Derek G. Corneil. The graph isomorphism disease. *Journal of Graph Theory*, 1:339–363, 1977.

[19] Steven W. Reyner. An analysis of a good algorithm for the subtree problem. *SIAM J. Comput.*, 6(4):730–732, 1977.

[20] U. Schöning. Graph isomorphism is in the low hierarchy. *Journal. Comput. Syst. Science*, 37:312–323, 1987.

[21] Maciej M. Syslo. The subgraph isomorphism problem for outerplanar graphs. *Theoretical Computer Science*, 17:91–97, 1982.

[22] E. Ukkonen. On-line construction of suffix-trees. In *Algorithmica 14*, pages 249–260, 1995.

[23] P. Weiner. Linear pattern matching algorithms. In *Proc. of the 14th IEEE Symp. on Switching and Automata Theory*, pages 1–11, 1973.

[24] F. Frances Yao. Graph 2-isomorphism is NP-complete. *Information Processing Letters*, 9(2):68–72, August 1979.

Orthogonal and Quasi-upward Drawings with Vertices of Prescribed Size*
Extended Abstract

Giuseppe Di Battista, Walter Didimo, Maurizio Patrignani, and Maurizio Pizzonia

Dipartimento di Informatica e Automazione,
Università di Roma Tre, Rome, Italy
{gdb,didimo,patrigna,pizzonia}@dia.uniroma3.it

Abstract. We consider the problem of computing orthogonal drawings and quasi-upward drawings with vertices of prescribed size. For both types of drawings we present algorithms based on network flow techniques and show that the produced drawings are optimal within a wide class. Further, we present the results of an experimentation conducted on the algorithms that we propose for orthogonal drawings. The experiments show the effectiveness of the approach.

1 Introduction

Orthogonal drawings are extensively used in many application areas and several algorithms for constructing orthogonal drawings can be found in the literature [13,4,7,8,14,17,12,18]. A widely adopted approach to produce orthogonal drawings is the so called topology-shape-metrics approach [9], originally proposed in [3,2,19,20].

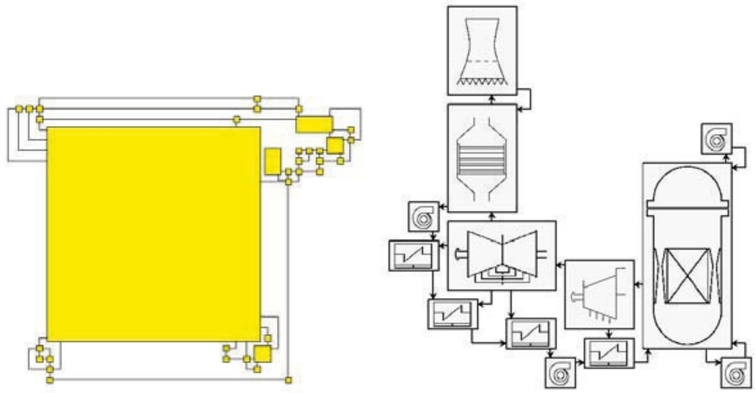
Although such approach has found in the last ten years several variations, implementations, and improvements, it is still unsuitable for producing drawings whose vertices have size assigned by the user. Actually, the algorithms based on the topology-shape-metrics approach make either the assumption that the vertices are points of the grid (see, e.g. [19]), or the assumption that all vertices have the same size (see, e.g. [2,13]).

However, orthogonal drawings with vertices of prescribed size have a wide range of applications. Examples are diagrams with long labels on symbols, diagrams for which the semantics or the importance of each vertex is related to its size, and diagrams whose vertices contain pictures or maybe other diagrams. Further, the capability of drawing vertices of prescribed size could be a key issue in the realization of new techniques for edges labelling. Namely, fictitious vertices of size suitable to host the labels could be inserted during a preprocessing step and replaced with the corresponding labels in the final drawing.

* Research supported in part by the CNR Project "Geometria Computazionale Robusta con Applicazioni alla Grafica ed al CAD", and by the ESPRIT LTR Project 20244 (ALCOM-IT)

J. Kratochvíl (Ed.): GD'99, LNCS 1731, pp. 297–310, 1999.

See in Fig. 1 two examples of drawings computed with the techniques described in this paper.



(a) (b)

Fig. 1. (a) A drawing with small and large vertices. (b) The basic blocks of an industrial plant (a boiling water reactor nuclear plant). Both drawings have been computed with an implementation of the techniques described in this paper.

In this paper we introduce in Section 2 a new drawing convention, called *podavsnef*, for planar orthogonal drawings with vertices of prescribed size. It is a variation of the *podevsnef* drawing convention presented in [13].

Second, we present in Section 3 an algorithm for constructing *podavsnef* drawings. The algorithm computes a *podevsnef* drawing and then expands the vertices. The expansion of vertices is shown to be optimal among a large set of possible expansions and the produced drawings can be further compacted with a post-processing compaction technique at the expenses of a higher computation time. Both the algorithm for vertex expansion and the post-processing are based on network flow techniques. The proposed algorithm was implemented and tested against the large test suite of graphs used in [10]. The height and the width of the vertices of the graphs have been randomly chosen in a wide interval. The experiments put in evidence the effectiveness of the techniques (Section 4).

Further, since the topology-shape-metrics approach has been recently extended to the construction of quasi-upward planar drawings of directed graphs [6], we also study the problem of producing quasi-upward planar drawings of digraphs with vertices of prescribed size (Section 5). We show an algorithm that is based on the construction of a visibility representation as an intermediate drawing.

2 Preliminaries and Drawing Conventions

We assume familiarity with planarity and connectivity of graphs [11,16] and with flow networks [1]. An *embedded planar graph* is a planar graph with a specified circular order of edges around vertices and a specified external face, admitting a planar drawing that respects the given embedding. Unless otherwise specified the planar graphs we consider are always embedded. A planar *st-digraph* G is an embedded planar digraph with only one source s and one sink t embedded on the external face. The *dual st*-digraph G^* of G is defined as follows. The faces of G are in one-to-one correspondence with the vertices of G^*, but for the external face of G that corresponds to two vertices s^* and t^* of G^*. For every edge e of G, G^* has an edge (f, g), where f is the face to the left of e and g is the face to the right of e. Digraph G^* may have multiple edges.

A *planar orthogonal drawing* of a planar graph is a planar drawing that maps each edge into a chain of horizontal and vertical segments. A *planar orthogonal grid drawing* is an orthogonal drawing such that vertices and bends along the edges have integer coordinates. A planar graph admits a planar orthogonal (grid) drawing if and only if its vertices have maximum degree four [21]. An *orthogonal representation* is an equivalence class of planar orthogonal drawings such that all the drawings of the class have the same sequence of left and right turns (bends) along the edges and two edges incident at a common vertex determine the same angle.

In order to orthogonally draw graphs of arbitrary vertex degree, different drawing conventions have been introduced. The *podevsnef* (planar orthogonal drawing with equal vertex size and not empty faces) drawing convention was introduced in [13]. In a *podevsnef* drawing (see Fig. 2.a):

1. Vertices are points of the grid but it is easier to think to them in terms of squares of half unit sides centered at grid points.
2. Two segments that are incident on the same vertex may overlap. Observe that the angle between such segments has zero degree.
3. All the polygons representing the faces have area strictly greater than zero.
4. If two segments overlap they are presented to the user as two very near segments.

In [13] an algorithm is presented that computes a *podevsnef* drawing of a planar graph with the minimum number of bends. Further, the authors conjecture that the drawing problem becomes NP-hard when Condition 3 is omitted. The *podevsnef* drawings generalize the concept of orthogonal representation, allowing angles between two edges incident to the same vertex to have a zero degree value. The consequence of the assumption that the polygons representing the faces have area strictly greater than zero is that the angles have specific constraints. Namely, because of Conditions 2 and 3, each zero degrees corresponds to exactly one bend [13]. An orthogonal representation corresponding to the above definition is a *podevsnef orthogonal representation.*

We also consider a similar drawing convention introduced in [5] called *simple-podevsnef*. In that model, the following constraints are added to the *podevsnef*

drawing convention: (i) In order to distribute the edges around a vertex more uniformly, each vertex with degree greater than four has at least one incident edge on each side. (ii) Consider a vertex u with $\deg(u) > 4$. Consider two edges (u, v) and (u, w) incident on u and such that (u, w) follows (u, v) in the circular clockwise ordering given by the embedding. If there is a zero degree angle at u between (u, v) and (u, w), then (i) (u, w) contains at least one bend and (ii) the first bend of (u, w) encountered while following (u, w) from u to w causes a right turn.

The *simple-podevsnef* drawing convention has several advantages [5]: the computation of its orthogonal representation can be done with a standard minimum cost flow technique and the final orthogonal drawing can be computed with a standard compaction technique. Observe that a *simple-podevsnef* drawing is a *podevsnef* drawing.

We generalize the concept of *podevsnef* drawing by introducing the *podavsnef* convention. A *podavsnef* (planar orthogonal drawing with assigned vertex size and non-empty faces) drawing is an orthogonal drawing such that:

1. Each vertex is a box with its specific height and width (assigned to each single vertex by the user).
2. Two segments that are incident on the same vertex may overlap. Observe that the angle between such segments has zero degree.
3. Consider any side of length $l \geq 0$ of a vertex v and consider the set I of arcs that are incident on such side.
   a) If $l + 1 > |I|$ then the edges of I cannot overlap.
   b) Else ($l + 1 \leq |I|$). The edges of I are partioned into $l + 1$ non-empty subsets such that all the edges of the same subset overlap.
4. The orthogonal representation constructed from a *podavsnef* drawing by contracting each vertex into a single point is a *podevsnef* orthogonal representation.

From the above definition it follows that a *podevsnef* drawing is a *podavsnef* drawing such that all its vertices have width and height both equal to zero. On the other hand a *podavsnef* drawing is essentially a *podevsnef* drawing where vertices have specific sizes and where the edges incident on each vertex side are uniformly distributed.

A *quasi-upward drawing* of a digraph [6] is such that the horizontal line through each vertex v (that is drawn as a point) "locally" splits the incoming edges of v from the outgoing edges of v. The term locally is used to identify a sufficiently small connected region properly containing v.

A *pqudavs* (planar quasi-upward drawing with assigned vertex size) drawing (Fig. 6.e) is a quasi-upward drawing such that each vertex is a box with its specific height and width (assigned to each single vertex by the user).

3 Computing *Podavsnef* Drawings

In this section we first show that, given a planar graph and an assignment of height and width for each of its vertices it is always possible to compute a

podavsnef drawing with the prescribed dimensions for vertices. Second, we show a complete strategy for constructing *podavsnef* drawings that allows trade-offs between effectiveness and efficiency.

Given a *podavsnef* drawing Γ and two vertical (horizontal) lines that do not intersect any vertex, a *vertical strip* (*horizontal strip*) is the set of the vertices of Γ contained in the geometric strip defined by the two lines. A *vertical partition* (*horizontal partition*) of Γ is the partition of the vertices of Γ into vertical (horizontal) strips with the maximum number of strips. We number the strips of the partition left to right (top to bottom).

A *podavsnef* drawing Γ is *splittable* if a vertical and a horizontal partition of the vertices of Γ exists such that: (1) A vertex v of Γ uniquely determines one horizontal strip $\sigma_H(i)$ and one vertical strip $\sigma_V(j)$ (we associate to v the pair i, j); and (2) The function associating a pair i, j to each vertex v is injective. The two partitions are a *split* of Γ. Observe that if Γ is splittable, then its split is unique.

Consider two *podavsnef* drawings Γ' and Γ'' of the same graph and with the same *podevsnef* orthogonal representation and with splits σ' and σ'', respectively. Splits σ' and σ'' are *equivalent* if for each vertex v, the pair i, j determined by v in σ' is the same v determines in σ''.

Given a planar graph G we construct a *podevsnef* drawing Γ of G, using one of the algorithms presented in [13,5]. Roughly speaking, our strategy consists of expanding the vertices inside different strips independently, preserving the shape of the drawing. After the expansion the strips are "glued" together.

Consider a *podevsnef* drawing Γ. Observe that Γ is always splittable. The split of Γ can be constructed as follows. The i-th horizontal (vertical) strip $\sigma_H(i)$ ($\sigma_V(i)$) is obtained considering the vertices of Γ with y-coordinates (x-coordinates) in the interval $[i - 1/2, i + 1/2]$. See Fig. 2.a.

We associate with a vertical strip $\sigma_V(i)$ a flow network $\mathcal{N}_i$ as follows (See Fig. 2.b). In $\sigma_V(i)$ a vertex u is *up-visible* from a vertex v if $y(u) > y(v)$ and a vertex x does not exist such that $y(u) > y(x) > y(v)$. If u is up-visible from v and u and v are joined by a straight edge, then v is *up-adjacent* to u.

The nodes of $\mathcal{N}_i$ are: Three nodes n^l_v, n^c_v, and n^r_v, for each vertex v of $\sigma_V(i)$. Two nodes $n^l_{u,v}$ and $n^r_{u,v}$ for each vertex u that is up-adjacent to a vertex v. One node $n^c_{u,v}$ for each vertex v that is up-visible from a vertex u that is not up-adjacent to v. One *source-node* s_i and one *sink-node* t_i.

For each vertex u that is up-adjacent to a vertex v we introduce the arcs $(n^c_u, n^r_{u,v})$, $(n^c_u, n^l_{u,v})$, $(n^r_{u,v}, n^c_v)$, $(n^l_{u,v}, n^c_v)$, $(n^l_u, n^l_{u,v})$, $(n^r_u, n^r_{u,v})$, $(n^l_{u,v}, n^l_v)$, and $(n^r_{u,v}, n^r_v)$.

For each vertex v that is up-visible from a vertex u that is not up-adjacent to v we introduce the arcs $(n^c_u, n^c_{u,v})$, $(n^c_{u,v}, n^c_v)$, $(n^l_u, n^c_{u,v})$, $(n^r_u, n^c_{u,v})$, $(n^c_{u,v}, n^l_v)$, and $(n^c_{u,v}, n^r_r)$.

Let u be the bottommost (topmost) vertex of $\sigma_V(i)$. We introduce the arcs (s_i, n^l_u), (s_i, n^c_u), and (s_i, n^r_u) ((n^l_u, t_i), (n^c_u, t_i), and (n^r_u, t_i)).

The units of flow correspond to units of width of the strip. We denote by $lb(\cdot)$ and $ub(\cdot)$ lower and upper bounds, respectively. Each node n^c_u has $lb(n^c_u) =$

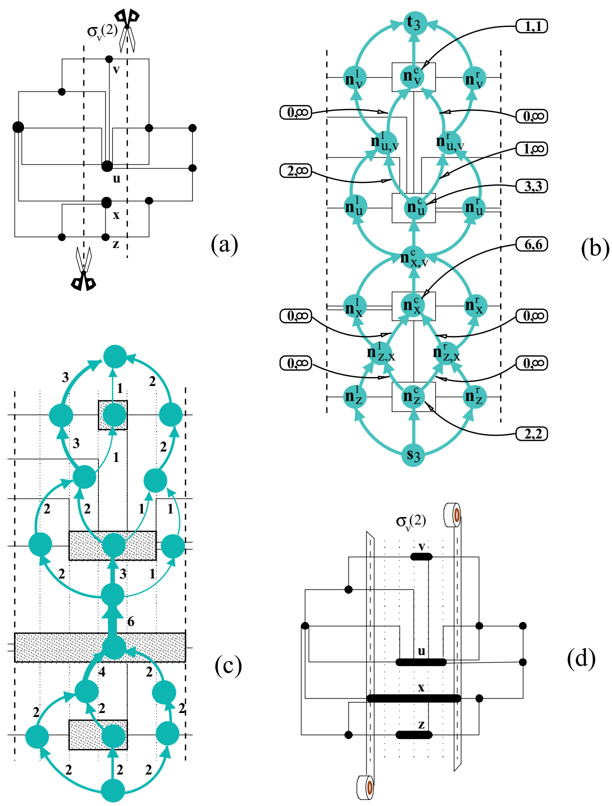


Fig. 2. (a) A *podevsnef* drawing and its vertical strip $\sigma_V(2)$. (b) The flow network $\mathcal{N}_2$. Labels show the most important lower and upper bounds. (c) A minimum flow in network $\mathcal{N}_2$. The thickness of the arcs is proportional to their flow; arcs and nodes with 0 flow are omitted. (d) Widths and positions of vertices in $\sigma_V(2)$.

$ub(n_u^c) = w \geq 0$, where w is the width assigned to u. The lower bounds of arcs $(n_{u,v}^l, n_v^c)$ and $(n_{u,v}^r, n_v^c)$ are used to preserve enough space for the edges incident on v from below. Namely, consider the sets I^l and I^r of arcs incident from below to the left and to the right of the straight edge connecting u to v, respectively. If $w + 1 > |I^l| + |I^r|$ then, according to the *podavsnef* convention, the incident edges can not overlap. Hence, we assign $lb(n_{u,v}^l, n_v^c) = |I^l|$ and $lb(n_{u,v}^r, n_v^c) = |I^r|$. Else ($w + 1 \leq |I^l| + |I^r|$), w is not enough large to draw the incident edges all non-overlapping. In this case we assign $lb(n_{u,v}^l, n_v^c) = 0$, $ub(n_{u,v}^l, n_v^c) = |I^l|$, $lb(n_{u,v}^r, n_v^c) = 0$, and $ub(n_{u,v}^r, n_v^c) = |I^r|$. Analogous bounds are assigned to $(n_u^c, n_{u,v}^l)$ and $(n_u^c, n_{u,v}^r)$. Lower and upper bounds not specified are set to 0 and to ∞, respectively. All arcs have an ∞ capacity and 0 cost. See Fig. 2.

Property 1. Network $\mathcal{N}_i$ is a planar *st*-digraph.

We define a network $\mathcal{N}_V$ associated with the vertical partition of Γ, obtained by the networks $\mathcal{N}_i$ by adding a new source s_V, a new sink t_V, and, for each i, the edges (s_V, s_i) and (t_i, t_V) ($lb(s_V, s_i) = lb(t_i, t_V) = 0$ and $ub(s_V, s_i) = ub(t_i, t_V) = \infty$). We apply an analogous construction to the horizontal partition of Γ, defining a network $\mathcal{N}_H$.

Property 2. Networks $\mathcal{N}_V$ and $\mathcal{N}_H$ are planar *st*-digraphs with $O(n)$ nodes, where n is the number of vertices of G.

From the above discussion we have:

Lemma 1. *Each pair of feasible flows one of $\mathcal{N}_V$ and the other of $\mathcal{N}_H$ determines a* podavsnef *drawing that admits a split equivalent to the one of Γ. Conversely, for each* podavsnef *drawing admitting a split equivalent to the one of Γ there exists a pair of feasible flows one of $\mathcal{N}_V$ and the other of $\mathcal{N}_H$.*

It is easy to see that networks $\mathcal{N}_V$ and $\mathcal{N}_H$ always have a feasible flow.

Because of Lemma 1 and since a *podevsnef* drawing of a planar graph always exists [13], we have:

Lemma 2. *Let G be a planar graph and suppose an assignment of widths and heights is given for the vertices of G. A* podavsnef *drawing of G always exists with vertices of the given sides.*

The values of flow in $\mathcal{N}_V$ and $\mathcal{N}_H$ are in one-to-one correspondence with the width and the height of the corresponding *podavsnef* drawings. A minimum width and height drawing can be computed by computing minimum flows on $\mathcal{N}_V$ and $\mathcal{N}_H$. Hence, we have:

Theorem 1. *Let G be a planar graph and let Γ be a* podevsnef *drawing of G. Suppose an assignment of widths and heights is given for the vertices of G. A* podavsnef *drawing of G can be computed that has minimum width and minimum height among those admitting a split that is equivalent to the one of Γ.*

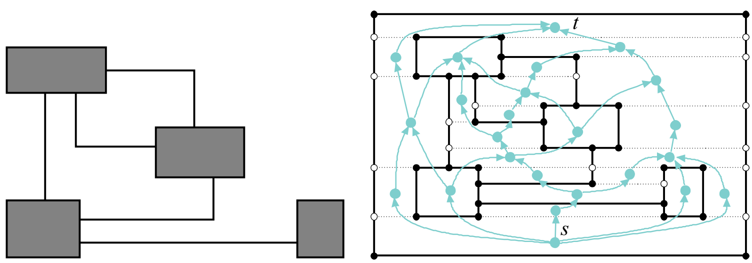
Observe that a minimum flow in $\mathcal{N}_V$ can be computed by solving a min-cost-flow problem on $\mathcal{N}_V$ augmented with arc (s_V, t_V) (with cost equal to zero) and by setting a cost equal to one on arcs (s_V, s_i). In such network we can "pump" any feasible flow of $\mathcal{N}_V$. The minimum flow in $\mathcal{N}_V$ is obtained by subtracting from the feasible flow the value of flow through (s_V, t_V).

It is clear that, even if the drawings produced with the above strategy are minimal in their equivalence class, they can be often further reduced in size if we allow vertices of a strips (vertical or horizontal) to enter another strip. This is not possible in the above framework.

If the user is interested in smaller drawings, even at the expenses of a higher computational time, a more effective technique can be adopted. The idea is to use the *podavsnef* drawing produced so far as a starting point for successive compaction steps.

We adopt a heuristic similar to the one used in the VLSI compaction [15], and that generalizes the "moving" technique presented in [12]. Namely, at each step we "squeeze" as much as possible the drawing in one direction. Then, we do the same in the opposite direction. We continue alternating the two steps until the drawing cannot be further squeezed. The steps should not modify the size of vertices. In the following we describe the algorithm for compacting with respect to the horizontal direction. The vertical case is analogous.

First (preprocessing step), we reduce the problem of horizontally compacting the *podavsnef* drawing Γ to the problem of horizontally compacting a drawing Γ' with all zero size vertices and constraints on the length of a subset of its edges. It is worth noting that the techniques described hereunder can be used in any case an analogous reduction can be found. The drawing Γ' is obtained from Γ as follows (see Fig. 3). (i) Each box of Γ representing a vertex is replaced with a box-shaped cycle in which the corners of the box and the attach points of the edges are represented by zero size vertices. We call *vertex-boundary edges* the edges created in this way. (ii) Each bend is replaced by a zero size vertex. The whole drawing is enclosed in a box consisting of four additional vertices and edges, in order to make easy to handle the external face in the subsequent steps.



(a) (b)

Fig. 3. The starting *podavsnef* drawing Γ (a) and The flow network $\mathcal{N}_V$ (b).

From Γ' we construct a drawing $\Gamma^{\#}$ such that all the faces are rectangles. This is done by splitting the faces of Γ' with a suitable set of new horizontal edges. See for example Fig. 3.b. This operation requires, in general, the introduction of fictitious vertices (white colored in the figure).

We compute the compacted drawing with a flow technique. Namely, we determine the new lengths to be assigned to the horizontal edges by solving a min-cost-flow problem on a network $\mathcal{N}$ built as follows (see Fig. 3.b): (i) the nodes are the internal faces of $\Gamma^{\#}$; (ii) we insert an arc for each pair of vertically-adjacent faces, with the only exception of the adjacencies with the external face. In other words, the horizontal edges of $\Gamma^{\#}$ that are not on the external face are in one-to-one correspondence with the arcs. Each arc is directed from the bottom face to the upper face.

Each unit of flow corresponds to one unit of length. When the solution is computed the vertex coordinates can be easily assigned using a depth first search visit of the drawing. In order to preserve the size of vertices and to minimize the total length of the horizontal edges, we set bounds and costs on each arc a of $\mathcal{N}$ corresponding to edge e of $\Gamma^{\#}$ as follows: if e is a vertex-boundary edge, then we set $lb(a) = ub(a)$ equal to the value of the length of e. Since the value of the flow through a is fixed, its cost is irrelevant; if e belongs to $\Gamma^{\#}$ but it is not an edge of Γ', then $lb(a) = 1$, $ub(a) = \infty$, and the cost of a is set to 0; if e belongs to Γ' and it is not a vertex-boundary edge, then $lb(a) = 1$, $ub(a) = \infty$, and the cost of a is set to 1.

Property 3. Network $\mathcal{N}$ is a planar st-digraph with $O(n + b)$ nodes, where n is the number of vertices and b is the number of bends of Γ. The source s and the sink t of $\mathcal{N}$ are the bottom and the top faces of $\Gamma^{\#}$, respectively.

Network $\mathcal{N}$ admits a feasible flow. The value of flow produced by s is equal to the width of the box enclosing the drawing. It is easy to see that the solution of a min-cost-flow problem on $\mathcal{N}$ corresponds to a minimization of the total length of the horizontal edges.

It is also possible to exploit network $\mathcal{N}$ to minimize the total width of the drawing. This is done by using what we call the *pinch* technique. During the construction of Γ', before building the enclosing box we encapsulate Γ' into a rectangle whose horizontal sides are assigned a high cost ($lb = 0$ and $ub = \infty$).

It is worth noting that, even though the techniques described above considerably reduce the length of the edges, we still have space for improvements. In fact, the described procedure does not allow the attach points of the edges to "slide" along the boundaries of the vertices because the length of each vertex-boundary edge in Γ' is fixed. Such constraint can be relaxed.

In order to obtain the desired dimension for the vertices we can fix only the sum of the lengths of the vertex-boundary edges along the top and bottom sides of each vertex, leaving the attach points free to move along the boundary. Namely, the flow network $\mathcal{N}$ can be modified as follows (see Fig. 4.d):

- For each box-shaped face f of $\Gamma^{\#}$ representing a vertex v of Γ, we split the node of $\mathcal{N}$ associated with f into two nodes v_{out} and v_{in} and add a directed arc $e = (v_{in}, v_{out})$. The incoming edges of v become the incoming edges of v_{in} while the outgoing edges of v become the outgoing edges of v_{out}.
- We set $lb(e) = ub(e)$ equal to the width assigned to v, and cost of e equal to 0.
- For each vertex-boundary edge e' of f, we set the lower bound of the corresponding arc of $\mathcal{N}$ equal to 0 if e' is incident to a "corner" of f, and equal to 1 otherwise. The value of $ub(e')$ is set to ∞ and the cost of e' is set to 0.

The benefits deriving from the above modifications are put in evidence in Fig. 4.a, 4.b, and 4.c.

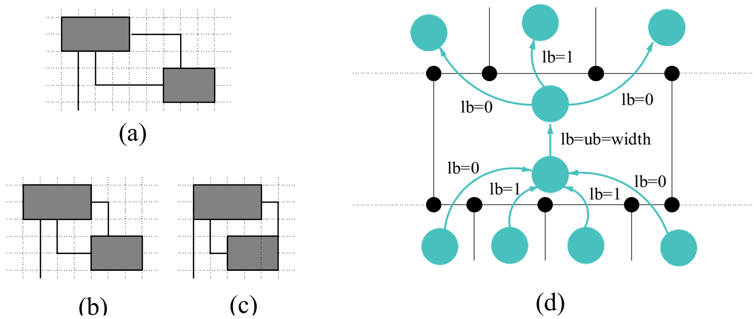


Fig. 4. Given an initial configuration (a), figures (b) and (c) show the results of the compaction without and with the refined strategy illustrated in figure (d).

4 Experiments with GDToolkit

We implemented the algorithms described in Section 3, that compute a *podavsnef* drawing from a *podevsnef* one. In particular our implementation constructs a *podavsnef* drawing starting from a *simple-podevsnef* drawing. The implementation uses the *GDToolkit* library[1].

We tested the algorithm over a set of more than 8000 (not-necessarily planar) graphs in the data-base used in [10], representing data from real-world applications. The number of vertices of the tested graphs is in the range $10 - -100$.

Our experimental setting is as follows. For each graph G in the test-suite we have randomly chosen the width w and the height h of every vertex v of G with a uniform probability distribution in the range $0 - 9$. The area covered by v (in terms of grid-points) is $(w+1)(h+1)$. Subsequently, we have computed a *simple-podevsnef* drawing Γ of G, and a *podavsnef* drawing Γ' of G starting from Γ. In the *simple-podevsnef* Γ all vertices have zero width and height, each one covering one unit of area. Both for Γ and for Γ' we measured the ratio between the total area covered by the vertices and the area of the drawing (both in terms of grid-points).

The experiments show that the compaction strategies described in the previous section make an effective usage of the area of the drawing, increasing the percentage of area used to represent vertices. Also, from the point of view of the CPU time, all the computations were performed within an acceptable amount of time. Namely, the largest graphs required less than 50 seconds on a PC Pentium II (350 MHz) with Linux and C++ code compiled with GNU g++.

5 Computing *Pqudavs* Drawings

In this section we describe how to compute a *pqudavs* drawing of a planar digraph. Since the strategy for computing a planar quasi-upward drawing relies on the

[1] `http://www.dia.uniroma3.it/~gdt`

construction of a planar upward drawing of a planar *st*-digraph derived from it [6], we focus on the problem of constructing a *pqudavs* drawing of a planar *st*-digraph. The techniques can be easily applied to draw general planar digraphs.

Planar upward drawings of planar *st*-digraphs can be constructed by using, as intermediate drawing, a visibility representation [9]. *Visibility representations* map vertices to horizontal segments (*vertex-segments*) and edges to vertical segments (*edge-segments*). The vertical segment representing edge (u, v) has its endpoints on the horizontal segments representing vertices u and v, and does not intersect with any other horizontal segment.

The basic technique [9] to compute coordinates of the vertex-segments of a visibility representation of an *st*-digraph G consists of computing two optimal topological numberings, one performed on the *st*-digraph G and the other performed on its dual *st*-digraph G^*. We recall that a *topological numbering* of an *st*-digraph is an assignment of integer numbers to its vertices, such that, for each edge (u, v), the number assigned to v is greater than the one assigned to u. The numbering is *optimal* if the range of numbers assigned to the vertices is minimized. The y-coordinate of the vertex-segment representing vertex v is the number assigned to v by the optimal topological numbering of G. The left and right x-coordinates of the vertex-segment representing vertex v are: (left x-coordinate) the lowest number assigned by the optimal topological numbering of G^* to the vertices of the face associated with v; (right x-coordinate) the highest number assigned by the optimal topological numbering of G^* to the vertices of the face associated with v minus one (See Fig. 5).

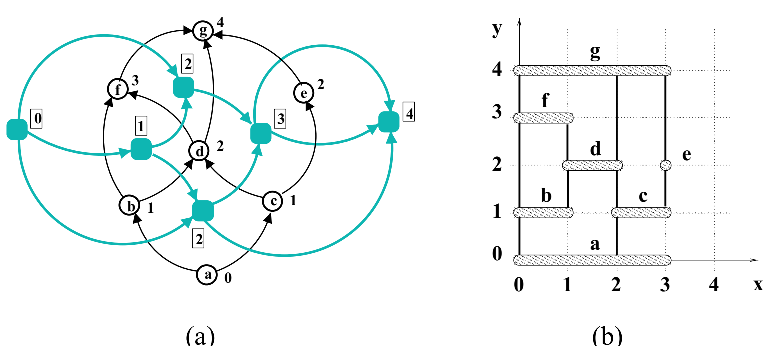


Fig. 5. (a) An *st*-digraph (colored black and drawn upward) and its dual (colored grey). Labels are the result of two optimal topological numberings. (b) The corresponding visibility representation.

Given an *st*-digraph G and an assignment of specific height and width to each of its vertices, we compute a *pqudavs* drawing of G by using, as intermediate drawing, an *expanded* visibility representation. In the expanded visibility representation a vertex with assigned height h and width w is represented by a box (*vertex-box*) of height h and width greater than or equal to w, while an

edge (u, v) is represented as usual by a vertical segment connecting the boxes representing vertices u and v, and not intersecting with any other vertex-box.

We assign the y-coordinates (x-coordinates) of the vertex-boxes of the expanded visibility representation by means of a network flow technique, computed on the flow network $\mathcal{N}_y$ ($\mathcal{N}_x$) defined as follows.

1. Starting from G, we construct a digraph G' (see Fig. 6.a) by replacing each vertex v of G with two new vertices v_{in} and v_{out} and with a new directed edge (v_{in}, v_{out}). The incoming edges of v become the incoming edges of v_{in} while the outgoing edges of v become the outgoing edges of v_{out}. Observe that G' is a planar st-digraph.
2. We compute the dual st-digraph G'^* of G'.
3. The nodes and arcs of $\mathcal{N}_y$ are (see Fig. 6.b) the vertices and edges of G'^*. Let v be a vertex of G with assigned height h. Let e be the edge of G'^* associated with (v_{in}, v_{out}). We set in $\mathcal{N}_y$ $lb(e) = ub(e) = h$. All the other arcs of $\mathcal{N}_y$ have lower bound equal to one and upper bound equal to ∞.
4. The nodes and arcs of $\mathcal{N}_x$ are the vertices and edges of G'. Let v be a vertex of G with assigned width w. We set in $\mathcal{N}_x$ $lb(v_{in}, v_{out}) = w + 1$. All the other arcs of $\mathcal{N}_x$ have lower bound equal to one. All the arcs have upper bound equal to ∞.

Each unit of flow on an arc of the network $\mathcal{N}_x$ corresponds to a unit of width required by the corresponding edge or vertex-box of G. Each unit of flow on an arc of the network $\mathcal{N}_y$ corresponds to a unit of height of the corresponding edge or vertex-box of G. Therefore, from a pair of feasible flows of $\mathcal{N}_y$ and $\mathcal{N}_x$ the coordinates of the vertex-boxes and of the edge-segments of the expanded visibility representation are easily computed (see Fig. 6.c and 6.d).

Property 4. Networks $\mathcal{N}_y$ and $\mathcal{N}_x$ are planar st-digraphs with $O(n)$ nodes, where n is the number of vertices of G.

Observe that, since all the upper bounds on the arcs of $\mathcal{N}_x$ are set to ∞, there always exists a feasible flow in such network. Further, in the network $\mathcal{N}_y$ the only arcs with a bounded upper capacity are those corresponding to the edges of G' derived from the splitting of vertices. However, for each of these arcs there is a directed path connecting its end vertices, composed by arcs with infinite upper bound. Then ([1]) there always exists a feasible flow in $\mathcal{N}_y$.

Since networks $\mathcal{N}_y$ and $\mathcal{N}_x$ always have a feasible flow, and since from an expanded visibility representation a *pqudavs* drawing can be easily computed with the techniques illustrated in [9] and in [6], we have:

Theorem 2. *Let G be a quasi-upward planar digraph and suppose an assignment of widths and heights is given for the vertices of G. A* pqudavs *drawing of G always exists.*

The amounts of flow in $\mathcal{N}_y$ and $\mathcal{N}_x$ are in one-to-one correspondence with the height and width of the expanded visibility drawings. Hence, to construct drawings with limited width and height we can compute minimum flows on $\mathcal{N}_y$ and

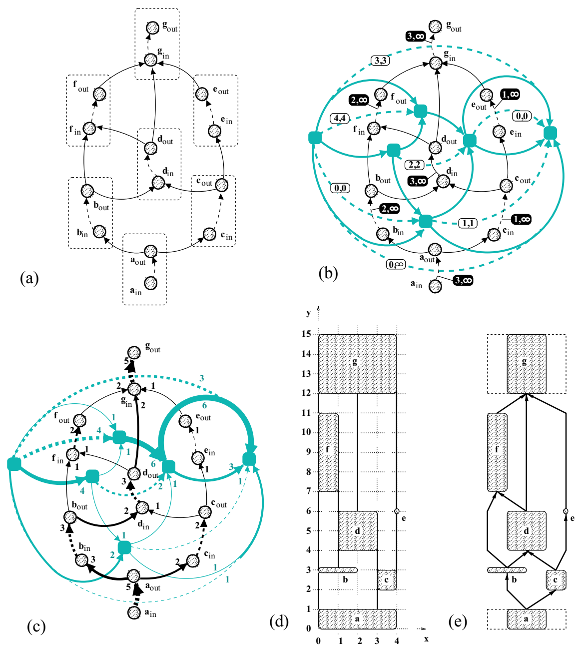


Fig. 6. (a) The digraph G' obtained by splitting each vertex of the input graph G (the digraph of Fig. 5). (b) The network flows $\mathcal{N}_x$ (black arcs) and $\mathcal{N}_y$ (grey arcs). Black (white) labels show the most important lower and upper bounds of $\mathcal{N}_x$ ($\mathcal{N}_y$). Unspecified lower bounds and upper bounds are assumed to be 1 and ∞, respectively. The lower and upper bounds on dashed arcs are used to specify the vertex dimensions. (c) Two minimum flows for $\mathcal{N}_x$ and $\mathcal{N}_y$. The thickness of the arcs is proportional to their flow. Arcs with flow 0 are omitted. (d) The expanded visibility representation of G corresponding to the flow values in (c). (e) A *pqudavs* drawing of the digraph G with the vertices of the specified height and width.

$\mathcal{N}_x$. A further level of optimization of the drawing can be reached by assigning one unit of cost to the arcs of $\mathcal{N}_y$ and performing a min-cost-flow on $\mathcal{N}_y$ with the above minimum flow as feasible flow. Since units of flow represent lengths of the edge-segments, this technique allows to obtain drawings with reduced total edge length. Unfortunately, since there are many degrees of freedom in obtaining the *st*-digraph G from the input digraph, the reduced size of the drawing of the *st*-digraph does not imply the reduced size of the *pqudavs* drawing of the input graph.

References

1. R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms, and Applications.* Prentice Hall, Englewood Cliffs, NJ, 1993.
2. C. Batini, E. Nardelli, and R. Tamassia. A layout algorithm for data flow diagrams. *IEEE Trans. Softw. Eng.*, SE-12(4):538–546, 1986.

3. C. Batini, M. Talamo, and R. Tamassia. Computer aided layout of entity-relationship diagrams. *Journal of Systems and Software*, 4:163–173, 1984.
4. P. Bertolazzi, G. Di Battista, and W. Didimo. Computing orthogonal drawings with the minimum number of bends. In F. Dehne, A. Rau-Chaplin, J.-R. Sack, and R. Tamassia, editors, *Proc. 5th Workshop Algorithms Data Struct.*, volume 1272 of *Lecture Notes Comput. Sci.*, pages 331–344. Springer-Verlag, 1997.
5. P. Bertolazzi, G. Di Battista, and W. Didimo. Computing orthogonal drawings with the minimum number of bends. manuscript, 1998.
6. P. Bertolazzi, G. Di Battista, and W. Didimo. Quasi-upward planarity. In S. H. Whitesides, editor, *Graph Drawing (Proc. GD '98)*, volume 1547 of *Lecture Notes Comput. Sci.*, pages 15–29. Springer-Verlag, 1998.
7. T. Biedl and G. Kant. A better heuristic for orthogonal graph drawings. *Comput. Geom. Theory Appl.*, 9:159–180, 1998.
8. T. C. Biedl. Relating bends and size in orthogonal graph drawings. *Inform. Process. Lett.*, 65:111–115, 1998.
9. G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. *Graph Drawing.* Prentice Hall, Upper Saddle River, NJ, 1999.
10. G. Di Battista, A. Garg, G. Liotta, R. Tamassia, E. Tassinari, and F. Vargiu. An experimental comparison of four graph drawing algorithms. *Comput. Geom. Theory Appl.*, 7:303–325, 1997.
11. S. Even. *Graph Algorithms.* Computer Science Press, Potomac, Maryland, 1979.
12. U. Fößmeier, C. Hess, and M. Kaufmann. On improving orthogonal drawings: The 4M-algorithm. In S. Whitesides, editor, *Graph Drawing (Proc. GD '98)*, volume 1547 of *Lecture Notes Comput. Sci.*, pages 125–137. Springer-Verlag, 1999.
13. U. Fößmeier and M. Kaufmann. Drawing high degree graphs with low bend numbers. In F. J. Brandenburg, editor, *Graph Drawing (Proc. GD '95)*, volume 1027 of *Lecture Notes Comput. Sci.*, pages 254–266. Springer-Verlag, 1996.
14. U. Fößmeier and M. Kaufmann. Algorithms and area bounds for nonplanar orthogonal drawings. In G. Di Battista, editor, *Graph Drawing (Proc. GD '97)*, volume 1353 of *Lecture Notes Comput. Sci.*, pages 134–145. Springer-Verlag, 1997.
15. T. Lengauer. *Combinatorial Algorithms for Integrated Circuit Layout.* Wiley-Teubner, 1990.
16. T. Nishizeki and N. Chiba. Planar graphs: Theory and algorithms. *Ann. Discrete Math.*, 32, 1988.
17. A. Papakostas and I. G. Tollis. Algorithms for area-efficient orthogonal drawings. *Comput. Geom. Theory Appl.*, 9(1–2):83–110, 1998. Special Issue on Geometric Representations of Graphs, G. Di Battista and R. Tamassia, editors.
18. J. M. Six, K. G. Kakoulis, and I. G. Tollis. Refinement of orthogonal graph drawings. In S. Whitesides, editor, *Graph Drawing (Proc. GD '98)*, volume 1547 of *Lecture Notes Comput. Sci.*, pages 302–315. Springer-Verlag, 1999.
19. R. Tamassia. On embedding a graph in the grid with the minimum number of bends. *SIAM J. Comput.*, 16(3):421–444, 1987.
20. R. Tamassia, G. Di Battista, and C. Batini. Automatic graph drawing and readability of diagrams. *IEEE Trans. Syst. Man Cybern.*, SMC-18(1):61–79, 1988.
21. L. Valiant. Universality considerations in VLSI circuits. *IEEE Trans. Comput.*, C-30(2):135–140, 1981.

Multi-dimensional Orthogonal Graph Drawing with Small Boxes

Extended Abstract

David R. Wood

School of Computer Science and Software Engineering
Monash University
Melbourne, Australia
davidw@csse.monash.edu.au

Abstract. In this paper we investigate the general position model for the drawing of arbitrary degree graphs in the D-dimensional ($D \geq 2$) orthogonal grid. In this model no two vertices lie in the same grid hyperplane. We show that for $D \geq 3$, given an arbitrary layout and initial edge routing a crossing-free orthogonal drawing can be determined.
We distinguish two types of algorithms. Our *layout-based* algorithm, given an arbitrary fixed layout, determines a degree-restricted orthogonal drawing with each vertex having aspect ratio two. Using a *balanced* layout this algorithm establishes improved bounds on the size of vertices for 2-D and 3-D drawings. Our *routing-based* algorithm produces 2-degree-restricted 3-D orthogonal drawings.
One advantage of our approach in 3-D is that edges are typically routed on each face of a vertex; hence the produced drawings are more truly three-dimensional than those produced by some existing algorithms.

1 Introduction

In this paper we consider orthogonal drawings of an n-vertex m-edge simple graph $G = (V, E)$ with maximum degree Δ. The *D-dimensional orthogonal grid* ($D \geq 2$) is the D-dimensional cubic lattice, consisting of *grid points* with integer coordinates, together with the coordinate-axis-parallel *grid lines* determined by these points. An integer i, $1 \leq i \leq D$, is called a *dimension*, and an integer d, $1 \leq |d| \leq D$, is called a *direction* (with the obvious interpretation).

An *orthogonal drawing* of G represents vertices $v \in V$ by pairwise non-intersecting *boxes*; i.e. sets $\{(a_1, a_2, \ldots, a_D) : l_i(v) \leq a_i \leq r_i(v), 1 \leq i \leq D\}$ for some closed integer intervals $[l_i(v), r_i(v)]$, $1 \leq i \leq D$. The graph-theoretic term 'vertex' will also refer to the corresponding box[1]. The size of a vertex v in a D-dimensional orthogonal drawing is denoted by $\alpha_1(v) \times \cdots \times \alpha_D(v)$ where $\alpha_i(v) = r_i(v) - l_i(v) + 1$.

[1] Vertices are possibly degenerate; this is the approach taken in [3, 4, 23], but not in [18]; enlarging vertices to remove this degeneracy increases the volume by a multiplicative constant.

J. Kratochvíl (Ed.): GD'99, LNCS 1731, pp. 311–322, 1999.

For each direction d we call the face of a box extremal in direction d the *d-face*. At each grid point on the d-face of a box there is a *port* in direction d. The number of ports on a box will be called its *surface*, and we shall refer to the number of grid points in a box as its *volume* (and *area* in two dimensions).

Each edge is represented by a sequence of contiguous segments of grid lines called an *edge route* possibly bent at grid points, and only intersecting its incident vertices. Edge routes are pairwise non-overlapping, and only in the 2-D orthogonal grid are edge routes allowed to cross. An orthogonal drawing with no more than b bends per edge route is called a *b-bend orthogonal drawing*. The directed graph with vertex set V consisting of the *reversal* arcs vw, wv for each edge $vw \in E$ is denoted by $G' = (V, A(G))$. An orthogonal drawing of G assigns each arc $vw \in A(G)$ a unique port at v.

An orthogonal drawing with a particular shape of box representing every vertex, e.g. point, line, square, box or cube, will be called an orthogonal *shape*-drawing for each particular shape, as illustrated in Fig. 1.

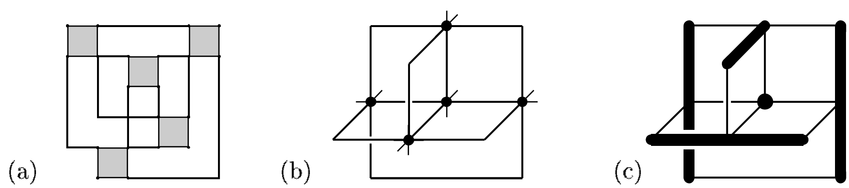


Fig. 1. Orthogonal drawings of K_5: (a) 1-bend 2-D square-drawing, (a) 2-bend 3-D point-drawing, (b) 0-bend 3-D line-drawing.

Orthogonal point-drawings have been studied extensively in two dimensions (see [7]) and to a lesser extent in three dimensions [10, 11, 18, 21]. However, D-dimensional point-drawings can only exist for graphs with maximum degree at most $2D$. Overcoming this restriction has motivated recent interest in 2-D box-drawings [1, 2, 6, 9, 12, 14, 15, 13, 17] and in 3-D box-drawings [3, 4, 18, 23]. In this paper we shall present unified results for orthogonal graph drawing in two and more dimensions.

The smallest D-dimensional box surrounding a D-dimensional orthogonal drawing is called the *bounding box*. The bounding box volume and the maximum number of bends per edge route are the most commonly proposed measures for determining the aesthetic quality of an orthogonal drawing. For box-drawings the shape and size of a vertex with respect to its degree are also considered an important measure of aesthetic quality. A vertex v is said to be *a-degree-restricted* if the surface$(v) \leq a \times \deg(v) + O(1)$. If for some constants a and d independent of the input graph, every vertex v with $\deg(v) \geq d$ is a-degree-restricted, then we say the drawing is *a-degree-restricted*. The *aspect ratio* of v is $\max_i \alpha_i(v) / \min_i \alpha_i(v)$. Degree-restricted orthogonal drawings with bounded aspect ratio are considered aesthetically pleasing.

Algorithms for 2-D orthogonal graph drawing which follow the so-called *Topology-Shape-Metrics* approach include the algorithms of Fößmeier, Kant and Kaufmann [13, 14, 15] in the *Kadinsky* model, the GIOTTO algorithm [8], and the recent algorithm of Di Battista et al. [6] which allows the user to specify vertex sizes. This approach based on network flow techniques can be traced to the classical algorithm of Tamassia [19] for finding a bend-minimum orthogonal point-drawing which preserves a fixed planar embedding.

Algorithms which do not guarantee crossing-free drawings, even for planar graphs, include that of Even and Granot [12], Papakostas and Tollis [17] and Biedl and Kaufmann [1]. The latter two algorithms both produce 2-degree-restricted 2-D orthogonal drawings with each vertex v having aspect ratio at most $\deg(v)/2$. At the expense of an increase in area, bounded aspect ratio drawings are produced by a second algorithm in [1] and by the *layout-based* algorithm presented in this paper, which also improves the degree restriction bound to 3/2. Using a diagonal layout our algorithm produces 2-degree-restricted square-drawings. Table 1 summarizes bounds for 2-D orthogonal graph drawing.

Table 1. Upper Bounds for 2-D Orthogonal Graph Drawing

area	max bends	degree restriction	aspect ratio	reference
$(m-1) \times (\frac{m+1}{2})$	1	2	$\deg(v)/2$	[17]
$(\frac{m+n}{2}) \times (\frac{m+n}{2})$	1	2	$\deg(v)/2$	[1]
$(\frac{3m+2n}{4}) \times (\frac{3m+2n}{4})$	1	2	2	[1]
$(\frac{3m+4n+2}{4}) \times (\frac{3m+4n+2}{4})$	1	3/2	2	Theorem 6
$\left(\frac{3(m+n)}{4}\right) \times \left(\frac{3(m+n)}{4}\right)$	1	2	1	Theorem 8

The trade-off between the maximum number of bends per edge route and the bounding box volume apparent in 3-D point-drawing algorithms [10, 11], is seen to a lesser extent in 3-D orthogonal box-drawings. Biedl et al. [3] constructs 1-, 2- and 3-bend 3-D orthogonal drawings of K_n with respective bounding box volumes $O(n^3)$, $O(n^3)$ and $O\left(n^{5/2}\right)$. However for arbitrary graphs the drawings are not necessarily degree-restricted.

Papakostas and Tollis [18] first established that every graph has a degree-restricted 3-D orthogonal drawing. Their bounding box volume upper bound has subsequently been improved to $O(n^3)$ by the *lifting half-edges* line-drawing algorithm of Biedl [4]. At the cost of an increase in volume, a modified technique is used to produce cube-drawings. Biedl also presents algorithms for 3-D line- and cube-drawings in general position, which is the model employed in this paper.

In [23] an algorithm which generalizes the COMPACT point-drawing algorithm of Eades et al. [10, 11] produces 6-bend 3-D orthogonal line-drawings of

multigraphs with bounding box volume $O(m^2/\sqrt{n})$, which is the best known bounding box volume upper bound for graphs with $O(n^{7/4})$ edges.

Our *layout-based* algorithm improves the best known bound for the degree-restriction of vertices in bounded aspect ratio 3-D orthogonal drawings. Using a diagonal vertex layout the algorithm produces cube-drawings. Our *routing-based* algorithm produces 2-degree-restricted orthogonal drawings. Table 2 summarizes the bounds for 3-D orthogonal graph drawings.

Table 2. Upper Bounds for 3-D Degree-Restricted Orthogonal Drawings

max bends	volume	degree restriction	aspect ratio	method
2	$O(m^3)$	6	?	incremental [18]
2	$O(n^3)$	2	$\deg(v)/2$	lifting $\frac{1}{2}$-edges [4]
2	$O(n^2 m)$	6	1	lifting $\frac{1}{2}$-edges [4]
2	$O(n^2 m)$	2	$\deg(v)/2$	routing-based [4]
2	$O\big((nm)^{3/2}\big)$	6	1	routing-based [4]
5	$O(m^2)$	2	$\deg(v)/2$	compact [23]
6	$O(m^2/\sqrt{n})$	2	$\deg(v)/2$	compact [23]
2	$O\big((nm)^{3/2}\big)$	10/3	2	layout-based (Theorem 7)
2	$O\big((nm)^{3/2}\big)$	4	1	layout-based (Theorem 8)
2	$O(n^2 m)$	2	$\deg(v)/2$	layout-based (Theorem 9)
2	$O(\Delta(nm)^{3/2})$	2	$\deg(v)/4$	routing-based (Theorem 10)

We shall present our algorithms in the following three stages (closely related to the so-called *three-phase method* [1, 2, 4]):

Vertex Layout: Determine the relative positions of the vertices.
Edge Routing: Determine the shape of each edge route.
Port Assignment: Construct vertex boxes and assign ports to arcs.

In Sec. 2 of this paper we introduce the general position model for orthogonal drawing and describe in detail the 'Port Assignment' stage of our algorithm. Our vertex layout methods are presented in Sec. 3. We distinguish two types of algorithms for the drawing of graphs in the general position model. Our *layout-based* method, described in Sec. 4, determines the edge routing with respect to a given vertex layout. This is the first algorithm for constructing orthogonal drawings in 2, 3 or more dimensions. We establish results for fixed, balanced and diagonal layouts. In Sec. 5 we discuss *routing-based* algorithms, where a layout is determined with respect to a given routing. See [22] for details of omitted proofs.

In a *vertex ordering* $(v_1, v_2, \ldots, v_n)$ of G, if $v_i v_j \in E$ $(i < j)$ we say v_j is a *successor* of v_i and v_i is a *predecessor* of v_j. The number of predecessors and successors of v_i are denoted $p(v_i)$ and $s(v_i)$ respectively. For directed graphs we only count the outgoing edges at v_i in $p(v_i)$ and $s(v_i)$.

2 The General Position Model

A D-dimensional orthogonal drawing is in *general position* if no grid hyperplane intersects any two vertices. This model has been used for 2-D orthogonal box-drawing in [1, 17], for 3-D point-drawing in [10, 11, 21] and for 3-D box-drawing in [4, 18, 22]. It is particularly useful in $D \geq 3$ dimensions since, as we shall prove, edge route intersections can always be eliminated.

The relative coordinates of the vertices in a general position D-dimensional orthogonal drawing of G are represented by D vertex orderings, called a *layout* of G. We write $v <_i w$, $1 \leq i \leq D$, if v is before w in the i-ordering. Suppose $V_i(v) = \{w : w <_i v\}$. Then v has a minimum i-coordinate of 0 if $V_i(v) = \emptyset$, and of $\sum_{w \in V_i(v)} \alpha_i(w)$ otherwise. We denote the number of successor and predecessors of v in the i-ordering by $s_i(v)$ and $p_i(v)$ respectively.

The assignment of ports to arcs is represented by a colouring of $A(G)$ with colours $\{1, 2, \ldots, D\}$ (or $\{X, Y\}$ and $\{X, Y, Z\}$ in 2 and 3 dimensions). An arc vw coloured i is assigned a port on the $(+i)$-face of v if $v <_i w$ and on the $(-i)$-face of v if $w <_i v$. The maximum of the number of arcs routed on the $(+i)$-face and on the $(-i)$-face of a vertex v is denoted $N_i(v)$.

All edge routes used by our algorithm have precisely $D-1$ bends, and thus for each edge vw, the ports assigned to the arcs vw and wv must be perpendicular; i.e. vw and wv are coloured differently. A colouring of $A(G)$ with this property is called a *routing* of G. Suppose the arcs vw and wv are respectively coloured i and j $(i < j)$. A $(D-1)$-bend edge route vw consists of consecutive grid-line segments between hyperplanes unique to v and w, respectively parallel to the following sequence of dimensions: $i \to (i-1) \to \ldots \to 1 \to (i+1) \to (i+2) \to \ldots \to (j-1) \to D \to (D-1) \to \ldots \to j$. We have the following upper bound for the volume of the bounding box.

Theorem 1. *A d-degree-restricted D-dimensional general position orthogonal drawing with each vertex having aspect ratio a has bounding box volume at most*

$$a \left(n^{D-2} \left(\frac{d}{D} m + \frac{O(1)}{2D} n \right) \right)^{D/(D-1)}$$

2.1 Determining Vertex Size

For each vertex v, we wish to determine positive integers $\alpha_i(v)$, $1 \leq i \leq D$, to minimize the surface(v); i.e.

$$\text{minimize} \sum_{i=1}^{D} \Bigg(\prod_{\substack{1 \leq j \leq D \\ j \neq i}} \alpha_j(v) \Bigg) \text{ such that } \forall i \prod_{\substack{1 \leq j \leq D \\ j \neq i}} \alpha_j(v) \geq N_i(v). \qquad (1)$$

A solution to (1) has surface $(v) \geq 2\sum_i N_i(v)$. We define κ_D to be the minimum k such that for every D-dimensional vertex v there is a solution to (1) with surface $(v) \leq k(2\sum_i N_i(v)) + O(1)$. A real-valued solution to (1) is given by

$$r_i(v) = \left(\Big(\prod_{\substack{1\leq j\leq D\\ j\neq i}} N_j(v)\Big) \Big/ N_i(v)^{D-2}\right)^{1/(D-1)} .$$

Lemma 1. *(a) $\kappa_2 = 1$, (b) $\kappa_3 \leq 2$ and (c) if $r_i(v) \geq 1$ for all i, $1 \leq i \leq D$, then $\kappa_D < 2^D$.*

Proof. (Outline) For $D = 2$ (1) is trivial: simply set $\alpha_X(v) = N_Y(v)$ and $\alpha_Y(v) = N_X(v)$. An integer-valued solution can obviously be obtained by setting $\alpha_i(v) = \lceil r_i(v)\rceil$. In the case of $D = 3$, if $r_i(v) \geq 2$ for all i, $1 \leq i \leq 3$, then this method determines a solution with $\kappa_3 \leq 2$. If for some i, $r_i(v) < 2$ then a case-by-case analysis establishes there is a solution with $\kappa_3 \leq 2$. For $D \geq 3$, if $r_i(v) \geq 1$ then $\alpha_i(v) < 2r_i(v)$ and $\kappa_D < 2^D$.

2.2 Port Assignment

For each face of a vertex we group the edges to be routed on this face according to the direction of their second segment. By the edge routing described in Sect. 2 there are four possible directions for the second segment. For each of the four groupings, we assign sufficiently many ports so that corresponding edges within different groups cannot intersect (see [22] for details). Within a grouping, ports are assigned to arcs vw in increasing order of the length of the first segment of the edge route from v to w, as illustrated in Fig. 2 (see [22] for details). on

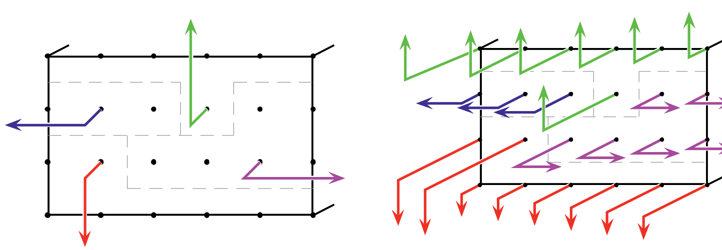


Fig. 2. Determining port assignments on a face.

Since a grid point on an edge route vw has at most one coordinate not unique to v or w, edge routes can only intersect if they are incident to a common vertex. For directions i and j, an (i,j)-*section* at a vertex v consists of the arcs vw where the first and second segments of the edge vw are in directions i and j and in the same ij-hyperplane. An edge vw is in exactly one section at v and one section at

w. Edge route intersections can only occur between edges in the same section. To eliminate edge crossings within an (i,j)-section (assume $i, j > 0$ – the other cases are easily inferred), choose the unrerouted arc vw such that w has maximum i coordinate (alternately with maximum j coordinate), and assign to vw the unassigned i-port in the section with minimum j-coordinate (i-coordinate), as illustrated in Fig. 3.

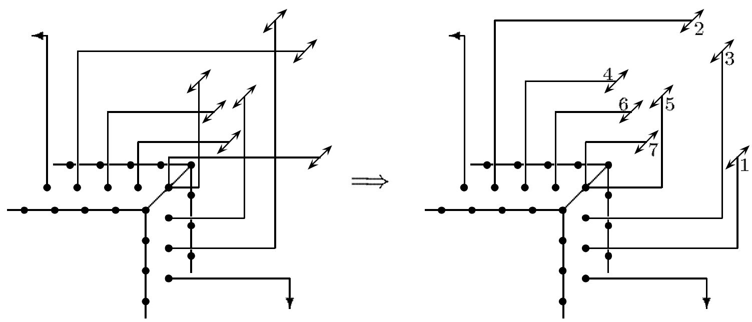


Fig. 3. Rerouting intersecting edge routes within a section in the order shown.

3 Balanced Graph Layout

For a given vertex ordering $(v_1, v_2, \ldots, v_n)$ of G, we say a vertex v is *positive* if $s(v) > p(v)$, *negative* if $p(v) > s(v)$ and *balanced* if $s(v) = p(v)$. For positive vertices v and for $k > 0$ (respectively, $k < 0$) v^k denotes the k^{th} successor (predecessor) of v to the right (left) of v in the ordering. For negative v and for $k > 0$ (respectively $k < 0$) v^k denotes the k^{th} predecessor (successor) of v to the left (right) of v in the ordering. Two adjacent vertices v_i and v_j $(i < j)$ are *opposite* if v_i is positive and v_j is negative. A vertex v has *cost* $c(v) = |s(v) - p(v)|$. We conjecture that determining a vertex ordering of a given graph with minimum total cost, the *balanced ordering problem*, is NP-hard.

We say a vertex ordering is *locally balanced* if moving any one vertex within the ordering does not reduce the total cost.

Lemma 2. *For every vertex v in a locally balanced vertex ordering, each vertex v^i, $1 \leq i \leq \lfloor c(v)/2 \rfloor$, is not opposite to v, as otherwise v could move past v^i and reduce the total cost.*

The following *median placement* heuristic for the balanced ordering problem will form the basis of our graph layout methods: Given a vertex ordering $(v_1, v_2, \ldots, v_n)$ of G, called the *insertion ordering*, for $i = 1, 2, \ldots, n$, insert v_i into the *current ordering* mid-way between its already inserted neighbours, i.e. between the predecessors of v_i in the insertion ordering.

Theorem 2. *The median placement algorithm determines in $O(m + n)$ time a vertex ordering of an undirected graph G with total cost at most $m + n$.*

Theorem 3. *For an acyclic (di)graph G, using a reverse topological ordering of G as the insertion ordering in the median placement heuristic, determines a vertex ordering of G with minimum total cost in $O(m+n)$ time.*

In a D-dimensional general position layout we define the cost of v to be the average cost of v over the D orderings. The following algorithm, based on a technique of [1], determines a 2-D *balanced* layout of a given graph: Arbitrarily order the vertices $(v_1, v_2, \dots, v_n)$. Determine the X- and Y-orderings using the median placement heuristic with insertion orderings $(v_1, v_2, \dots, v_n)$ and $(v_n, v_{n-1}, \dots, v_1)$ respectively. To determine a D-dimensional *balanced* layout calculate a 2-D balanced layout and set the i-ordering, $1 \le i \le D$, equal to the X/Y-ordering for odd/even i.

Theorem 4. *The above algorithm determines a D-dimensional general position layout in $O(D(m+n))$ time such that for each vertex v,*

$$c(v) \le 1 + \frac{\lceil D/2 \rceil}{D} \deg(v) \ .$$

This bound is tight within a small additive constant in the case of a D-dimensional layout of K_n if D is even. For odd D, it is an open problem to determine tight bounds for $\max_v c(v)$ in a D-dimensional layout.

4 Layout-Based Algorithms

We now we describe an algorithm which, given a D-dimensional layout of a graph G, determines a routing of G with bounds on the number of edges routed on each face. To represent the colouring of $A(G)$ we vertex-colour a graph $H = (A(G), E_H)$. So that reversal arcs receive different colours, for each edge vw of G, we add the edge $\{vw, wv\}$ to E_H, called an *r-edge*. For each vertex v and each orthant o relative to v, we partition the arcs $\{vw : w \in o\}$ into $\lceil |\{vw : w \in o\}| / D \rceil$ sets each of size at most D, and add a clique (consisting of *c-edges*) to H between vertices corresponding to arcs in the same partition, as in Fig. 4. The vertices of H corresponding to arcs in a partition with size less than D are said to be *leftover*. For $D = 2$, we add edges to H, called *l-edges*, between certain leftover vertices (see [22]).

Theorem 5. *Every fixed D-dimensional layout ($D \ge 2$) of G admits a degree-restricted orthogonal drawing of G that can be determined in $O(D(m+n))$ time such that:*

- *Each edge route has $D-1$ bends.*
- *The aspect ratio of each vertex tends to 2 (for large degree vertices).*
- *The bounding box volume is $O\left(n^{D(D-2)/(D-1)} m^{D/(D-1)}\right)$.*

In the case of $D = 2$ the vertices are 2-degree-restricted and the bounding box is at most $(m + 3n/4) \times (m + 3n/4)$. For $D = 3$ the vertices are 4-degree-restricted and the bounding box volume is at most $4\sqrt{2}(nm)^{3/2}$.

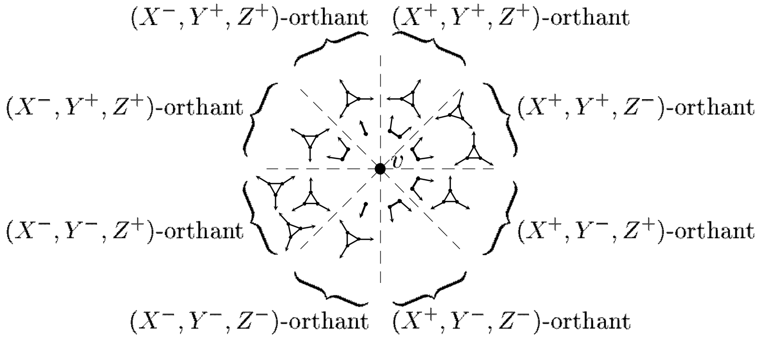


Fig. 4. Partitioning of $\{vw \in A(G)\}$ and construction of H for $D = 3$.

Proof. (Outline) For $D = 2$, a cycle in H consists of alternating r- and (c or l)-edges and is therefore of even length. Since $\Delta(H) \leq D$ and the complete graph $K_{D+1} \not\subseteq H$, by Brooks' Theorem, H is D-colourable in $O(|E_H|) = O(mD)$ time. The vertex-colouring of H determines a routing of G such that for each orthant o relative to a vertex v and in each partition of $\{vw : w \in o\}$, there is at most one arc vw coloured i, $1 \leq i \leq D$. It follows that,

$$N_i(v) \;\leq\; \frac{1}{D} \max\{s_i(v), p_i(v)\} + O(1). \tag{2}$$

Since $\deg(v)/2 \;\leq\; \max\{s_i(v), p_i(v)\} \;\leq\; \deg(v)$ our aspect bound follows. Also,

$$2\sum_{i=1}^{D} N_i(v) \;\leq\; \deg(v) + c(v) + O(1),$$

and hence

$$\text{surface}(v) \;\leq\; \kappa_D\,(\deg(v) + c(v)) + O(1).$$

Since $c(v) \leq \deg(v)$ and for sufficiently large $\deg(v)$, we have $r_i(v) \geq 1$, $1 \leq i \leq D$, it follows from Lemma 1(c) that $\kappa_D \leq 2^D$, and the drawing is degree-restricted. Since $\kappa_2 = 1$, a 2-D drawing is 2-degree-restricted. The area bound follows since

$$\sum_v \alpha_X(v) \;=\; \sum_v N_Y(v) \;\leq\; \frac{1}{4}\Big(3n + \sum_v (\deg(v) + c_Y(v))\Big) \;\leq\; \frac{3n}{4} + m.$$

By Lemma 1(b), $\kappa_3 = 2$, so a 3-D drawing is 4-degree-restricted. Our volume bounds follow from Theorem 1. □

Using the balanced layout algorithm described in Sec. 3 and the above routing algorithm we obtain the following results for 2-D and 3-D orthogonal drawings.

Theorem 6. *Every graph has a 2-D orthogonal drawing that can be determined in $O(m+n)$ time such that:*

- *Each edge route has* 1 *bend.*
- *Each vertex is* $\frac{3}{2}$*-degree-restricted.*
- *The aspect ratio of each vertex tends to 2 (for large degree vertices).*
- *The bounding box area is* $\left(\frac{3m+4n+2}{4}\right) \times \left(\frac{3m+4n+2}{4}\right)$.

Theorem 7. *Every graph has a 3-D orthogonal drawing that can be determined in $O(m+n)$ time such that:*

- *Each edge route has* 2 *bends.*
- *Each vertex is* $\frac{10}{3}$*-degree-restricted.*
- *The aspect ratio of each vertex tends to 2 (for large degree vertices).*
- *The bounding box volume is at most* $2.34(nm)^{3/2} + O(n^3)$.

Using a *diagonal* vertex layout with corresponding vertex ordering determined by the medium placement heuristic we obtain the following results.

Theorem 8. *Every graph has a D-dimensional degree-restricted hypercube-drawing ($D \geq 2$) which can be determined in $O(D(m+n))$ time such that:*

- *Each edge route has $D-1$ bends.*
- *The bounding box volume is at most* $\left((2n)^{D-2}\left(\frac{(2D-1)n+3m}{2D}\right)\right)^{D/(D-1)}$

Theorem 9. *Every graph has a D-dimensional line-drawing ($D \geq 3$) that can be determined in $O(D(m+n))$ time such that:*

- *Each edge route has $D-1$ bends.*
- *Each vertex is a* 2*-degree-restricted D-axis parallel line.*
- *The bounding box volume is at most* $n^{D-1}\left(\frac{(2D-3)n+3m}{2(D-1)}\right)$

5 Routing-Based Algorithms

Given a routing of G, we determine the i-ordering, $1 \leq i \leq D$, of a layout of G by applying the median placement heuristic to the subgraph of G', denoted $G'[i]$, induced by the arcs coloured i. If each $G'[i]$ is acyclic then we say the routing is *acyclic*, and by Theorem 3 minimum cost orderings can be determined.

To determine a 2-colour acyclic routing of G, start with a vertex ordering $(v_1, v_2, \ldots, v_n)$ of G, and for each edge $v_i v_j \in E$ ($i < j$) colour the arcs $v_i v_j$ X and $v_j v_i$ Y. Clearly, $G'[X]$ and $G'[Y]$ are both acyclic; the topological orderings of $G'[X]$ and $G'[Y]$ are respectively $(v_1, v_2, \ldots, v_n)$ and $(v_n, v_{n-1}, \ldots, v_1)$. This approach is used in [1] and in [4] for determining the routing and the X- and Y-orderings of a 3-D layout; each vertex is then represented by a line parallel to the Z-axis. The main criticism of this method is that the drawings are inherently two-dimensional.

We now describe a new method for determining a 3-colour acyclic routing. Firstly, determine a locally balanced vertex ordering $(v_1, v_2, \ldots, v_n)$ (see Sec. 3).

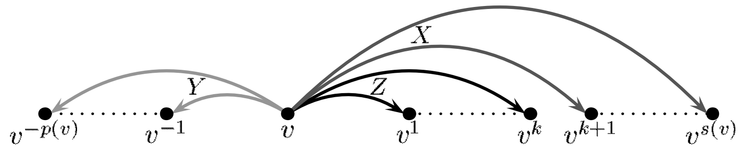


Fig. 5. Routing arcs at a positive vertex v; $k = \lfloor c(v)/2 \rfloor$.

For each vertex v_i, colour the arcs $v_i(v_i)^k$, $1 \leq k \leq \lfloor c(v_i)/2 \rfloor$, with colour Z. Remaining arcs $v_i v_j$ are coloured X if $i < j$ and Y if $j < i$, as in Fig. 5. Clearly $G'[X]$ and $G'[Y]$ are acyclic. By Lemma 2 a positive vertex v_i cannot have an incoming arc $v_j v_i \in G'[Z]$ with $i < j$. Similarly for negative vertices. Hence $G'[Z]$ is also acyclic.

Theorem 10. *Every graph has 3-D orthogonal drawing that can be determined in $O(m+n)$ time such that,*

- *Each edge route has* 2 *bends.*
- *Each vertex is 2-degree-restricted and has aspect ratio at most* $\deg(v)/4$.
- *The bounding box volume is* $O(\Delta(nm)^{3/2})$.

Proof. (Outline) For each vertex v, $2\sum_i N_i(v) = \deg(v) + O(1)$. Since $\kappa_3 = 2$, it follows that surface $(v) \leq 2\deg(v) + O(1)$, and v is 2-degree-restricted. A vertex v has maximum aspect ratio if, in the locally balanced vertex ordering, $c(v) = 0$, $s(v) = 0$ or $p(v) = 0$, in which case v is a line of length $\deg(v)/4$. Applying Theorem 1 we obtain a bounding box volume bound of $O(\Delta(nm)^{3/2})$. □

Acknowledgements

The author is grateful for the stimulating ideas of Graham Farr, and for useful discussions with the participants of AWOCA'97.

References

[1] T. Biedl and M. Kaufmann. Area-efficient static and incremental graph drawings. In R. Burkhard and G. Woeginger, editors, *Proc. Algorithms: 5th Annual European Symp. (ESA'97)*, volume 1284 of *Lecture Notes in Comput. Sci.*, pages 37–52, Berlin, 1997. Springer.

[2] T. Biedl, B. Madden, and I.G. Tollis. The three-phase method: A unified approach to orthogonal graph drawing. In Di Battista [5], pages 391–402.

[3] T. Biedl, T. Shermer, S. Whitesides, and S. Wismath. Orthogonal 3-D graph drawing. In Di Battista [5], pages 76–86.

[4] T.C. Biedl. Three approaches to 3D-orthogonal box-drawings. In Whitesides [20], pages 30–43.

[5] G. Di Battista, editor. *Proc. Graph Drawing: 5th International Symp. (GD'97)*, volume 1353 of *Lecture Notes in Comput. Sci.*, Berlin, 1998. Springer.

[6] G. Di Battista, W. Didimo, M. Patrignani, and M. Pizzonia. Orthogonal and quasi-upward drawings with vertices of arbitrary size. In J. Kratochvil, editor, *Proc. Graph Drawing: 7th International Symp. (GD'99)*, Lecture Notes in Comput. Sci., Berlin. Springer. to appear.

[7] G. Di Battista, P. Eades, R. Tamassia, and I.G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice-Hall, New Jersey, 1999.
[8] G. Di Battista, A. Garg, G. Liotta, R. Tamassia, E. Tassinari, and F. Vargiu. An experimental comparison of four graph drawing algorithms. *Comput. Geom.*, 7(5-6):303–325, 1997.
[9] W. Didimo and G. Liotta. Computing orthogonal drawings in a variable embedding setting. In K.-Y. Chwa and O. H. Ibarra, editors, *Proc. Algorithms and Computation (ISAAC'98)*, volume 1533 of *Lecture Notes in Comput. Sci.*, pages 79–88, Berlin, 1998. Springer.
[10] P. Eades, A. Symvonis, and S. Whitesides. Two algorithms for three dimensional orthogonal graph drawing. In North [16], pages 139–154.
[11] P. Eades, A. Symvonis, and S. Whitesides. Three dimensional orthogonal graph drawing algorithms. 1998. submitted.
[12] S. Even and G. Granot. Grid layouts of block diagrams — bounding the number of bends in each connection. In R. Tamassia and I.G. Tollis, editors, *Proc. Graph Drawing: DIMACS International Workshop (GD'94)*, volume 894 of *Lecture Notes in Comput. Sci.*, pages 64–75, Berlin, 1995. Springer.
[13] U. Fößmeier, G. Kant, and M. Kaufmann. 2-visibility drawings of planar graphs. In North [16], pages 155–158.
[14] U. Fößmeier and M. Kaufmann. Drawing high degree graphs with low bend numbers. In F.J. Brandenburg, editor, *Proc. Graph Drawing: Symp. on Graph Drawing (GD'95)*, volume 1027 of *Lecture Notes in Comput. Sci.*, pages 254–266, Berlin, 1996. Springer.
[15] U. Fößmeier and M. Kaufmann. Algorithms and area bounds for nonplanar orthogonal drawings. In Di Battista [5], pages 134–145.
[16] S. North, editor. *Proc. Graph Drawing: Symp. on Graph Drawing (GD'96)*, volume 1190 of *Lecture Notes in Comput. Sci.*, Berlin, 1997. Springer.
[17] A. Papakostas and I.G. Tollis. Orthogonal drawing of high degree graphs with small area and few bends. In F. Dehne, A. Rau-Chaplin, J.-R. Sack, and R. Tamassia, editors, *Proc. Algorithms and Data Structures: 5th International Workshop (WADS'97)*, volume 1272 of *Lecture Notes in Comput. Sci.*, pages 354–367, Berlin, 1997. Springer.
[18] A. Papakostas and I.G. Tollis. Incremental orthogonal graph drawing in three dimensions. In Di Battista [5], pages 52–63.
[19] R. Tamassia. On embedding a graph in the grid with the minimum number of bends. *SIAM J. Comput.*, 16(3):421–443, June 1987.
[20] S. Whitesides, editor. *Proc. Graph Drawing: 6th International Symp. (GD'98)*, volume 1547 of *Lecture Notes in Comput. Sci.* Springer, 1998.
[21] D.R. Wood. An algorithm for three-dimensional orthogonal graph drawing. In Whitesides [20], pages 332–346.
[22] D.R. Wood. Multi-dimensional orthogonal graph drawing in the general position model. Technical Report 99/38, School of Computer Science and Software Engineering, Monash University, Australia, 1999. (available at `http://www.csse.monash.edu.au/publications/`).
[23] D.R. Wood. A new algorithm and open problems in three-dimensional orthogonal graph drawing. In R. Raman and J. Simpson, editors, *Proc. Australasian Workshop on Combinatorial Algorithms (AWOCA'99)*, pages 157–167. Curtin University of Technology, Perth, 1999.

Geometric Realization of Simplicial Complexes

Patrice Ossona de Mendez[1]

CNRS UMR 8557 — E.H.E.S.S., 54 Bd Raspail, 75006 Paris, France
pom@ehess.fr

Abstract. We show that an abstract simplicial complex Δ may be realized on a grid of $\mathbb{R}^{d-1}$, where $d = \dim P(\Delta)$ is the order dimension (Dushnik-Miller dimension) of the face poset of Δ.

1 Introduction

Abstract simplicial complexes are related to order dimension in Section 2 through the complex of a d-representation. This construction is analogous to the one introduced by Dushnik in [10] and similar to the one used by Scarf in mathematical economy [14] under the name of *primitive sets* and which has been applied to integer programming by Barany, Howe and Scarf [1] and to commutative algebra by Bayer, Peeva and Sturmfels [2]. In Section 3, we give a simple necessary and sufficient condition for a mapping to be a geometric realization of an abstract simplicial complex. This characterization leads in Section 4 to the geometric realization of the abstract simplicial complex defined by a d-representation on a grid in $\mathbb{R}^{d-1}$. This generalizes the result of Schnyder on planar graphs [15] (see also [3][4]). In Section 5, we prove that any abstract simplicial complex may be triangulated into a "standard" d-representation having the same face poset dimension. In Section 6, we extend the vertex shelling order introduced earlier by Fraysseix, Pach and Pollack for planar triangulations [8] and show that a complex is vertex-shellable if and only if it is shellable (in the usual sense). We also prove that a vertex shelling order of the triangulation Δ^+ mentioned above may be easily derived from the "generating" total orders, generalizing a result proved in [5]. Eventually, Theorem 7 gathers most of the main results of the paper in a single statement.

We recall some basic definitions on simplicial complexes. For further information, see [12]. In the following, we will consider only finite simplicial complexes, what will justify the following definition. An *abstract simplicial complex* Δ is a non-empty finite collection of finite sets such that $X \in \Delta, Y \subseteq X$ implies $Y \in \Delta$. The union $V(\Delta)$ of the members of Δ is the *vertex set* of Δ. The members of Δ are the *faces* of Δ. The *dimension* of a face X of Δ is $\dim X = |X| - 1$. The *dimension* $\dim \Delta$ is the maximum dimension of any face of Δ. If Δ and Δ' are abstract simplicial complexes with disjoint ground sets, we recall that their *combinatorial join* is the abstract simplicial complex $\Delta * \Delta'$ defined by $\Delta*\Delta' = \{X \cup X', \quad X \in \Delta, X' \in \Delta'\}$. By extension, $\Delta * p$ will denote $\Delta * \{\emptyset, \{p\}\}$. An abstract simplicial complex Δ is *pure* if all the maximal faces of Δ (with respect to inclusion) have the same dimension, that is if any face of Δ is included in

J. Kratochvíl (Ed.): GD'99, LNCS 1731, pp. 323–332, 1999.

a face of Δ with dimension $\dim \Delta$. The *face poset* $P(\Delta)$ of Δ is the poset which consists of all the faces of Δ ordered by inclusion. For a poset P, a *realizer* of P is a set of total orders whose intersection is P. The minimum cardinality of a realizer of poset P is its *Dushnik-Miller dimension* (or simply the *dimension*) $\dim P$ [11].

2 The Complex of a d-Representation

Let $d > 0$ be an integer and let V be a finite set. A *d-representation* $R = (<_1, \dots, <_d)$ of V is a set of d total orders on V whose intersection is an antichain. With respect to R, the *supremum section* $S(X)$ of a subset $X \subseteq V$ is the subset of X whose elements are the maxima of X for some linear order in R:

$$S(X) = \{x \in X, \quad \exists 1 \leq i \leq d, \forall y \in X - \{x\}, x >_i y\} \tag{1}$$

Given a d-representation R of V, the *complex* of R is the set $\Sigma(R)$ of all the subsets X of V such that $\forall v \in V, v \in S(X \cup \{v\})$

Lemma 1. *The complex $\Sigma(R)$ of a d-representation R of V is an abstract simplicial complex.*

Proof. It is straightforward that $S(A) \supseteq S(B) \cap A$ whenever $A \subseteq B$. Thus, if X' is a subset of a set $X \in \Sigma(R)$, we get $v \in S(X' \cup \{v\})$ for any element $v \in V$ as $v \in S(X \cup \{v\})$. □

Theorem 1. *Let (V, Δ) be an abstract simplicial complex with vertex set V.*
Then, $\dim P(\Delta)$ is the smallest integer d, such that Δ is a subcomplex of some d-representation of V.

Proof.
- Δ is a subcomplex of a $\dim P(\Delta)$-representation of V:
Consider a realizer $\prec_1, \dots, \prec_d$ of cardinality $d = \dim P(\Delta)$ of $P(\Delta)$ and let $R = (<_1, \dots, <_d)$ be the d-representation of V induced by the restrictions on V of the d total orders $\prec_1, \dots, \prec_d$.
Let X be an element of Δ. Then, for all $1 \leq i \leq d$ and all $x \in X$ we have $x \prec_i X$, as X is, by definition, greater than its elements in the face poset. Moreover, X is not comparable to any element which does not belong to X. Hence, for any $y \notin X, \exists 1 \leq i \leq d, \forall x \in X, \quad x \prec_i y$. Hence, $\forall y \notin X, y \in S(X + y)$. Similarly, if x belongs to X, either $X = \{x\}$ and $x \in S(X)$ or $X - x$ is a simplex which belongs to Δ and hence $x \in S(X - x + x)$ and thus $x \in S(X)$. Altogether, X belongs to $\Sigma(R)$.
- If Δ is a subcomplex of a d-representation of V, then $d \geq \dim P(\Delta)$:
Insert in the d total orders of the representation the faces of Δ (different from vertices) the following way: In the linear order $<_i$, insert just after the vertex x all the faces including x and vertices smaller than x (with respect to linear order $<_i$), sorted by increasing size and, for a same size, in lexicographic order (with respect to $<_i$). Then, the face-inclusion of X in Y

in Δ obviously correspond to $X <_i Y$ (for all $1 \leq i \leq d$). Otherwise, if X and Y are not compared in $P(\Delta)$, there exists $a \in X \setminus Y$ and $b \in Y \setminus X$. As Δ is a subcomplex of the d-representation, there exists a linear order i such that a is greater than all the vertices of Y with respect to $<_i$. Hence, $X >_i Y$. Similarly, there exists j such that $Y >_j X$. Thus X and Y are not comparable in the intersection of the d total orders. Altogether, $P(\Delta)$ is equal to the intersection of the d total orders and hence $\dim P(\Delta) \leq d$.

□

3 Geometric Realizations of Simplicial Complexes

Consider an injective mapping $f : V(\Delta) \to \mathbb{R}^n$ (this mapping is naturally extended to map the subsets of $V(\Delta)$ to the corresponding subsets of $\mathbb{R}^n$). We shall say that f is a *geometric realization* of Δ in $\mathbb{R}^n$ if $f(\Delta)$ is a *geometric simplicial complex*, i.e. if

– for any face X of Δ, $f(X)$ is a set of affinely independent points (i.e. defines a simplex) of $\mathbb{R}^n$,
– the intersection of two faces of $f(\Delta)$ is a face of $f(\Delta)$, that is, for any faces X and Y of Δ:

$$\mathrm{Conv}(f(X)) \cap \mathrm{Conv}(f(Y)) = \mathrm{Conv}(f(X) \cap f(Y)) \tag{2}$$

where $\mathrm{Conv}(P)$ denotes the convex hull of the point set P.

If $V(\Delta)$ is a set of points in $\mathbb{R}^n$, Δ is thus a geometric simplicial complex if and only if the identity is a geometric realization of Δ. We shall say that an abstract simplicial complex Δ is *realizable* in $\mathbb{R}^n$ if there exists a geometric realization of Δ in $\mathbb{R}^n$. We remark that the first of the two conditions we gave for f to be a geometric realization implies $n \geq \dim \Delta$.

Lemma 2. *If* $\mathrm{Conv}(f(X)) \cap \mathrm{Conv}(f(Y)) = \mathrm{Conv}(f(X) \cap f(Y))$ *holds for any two faces X and Y of an abstract simplicial complex Δ, then f is a geometric realization of Δ.*

Proof. The injectivity of f is straightforward as $\mathrm{Conv}(f(\{x\})) \cap \mathrm{Conv}(f(\{y\})) = \emptyset$ whenever $x \neq y$. Thus, we only have to prove that the image of a set $X \in \Delta$ is a set of affinely independent points. Let us prove it *ad absurdum*. Assume $x_1, \ldots, x_k$ are elements of X having linearly dependent images by f. Up to a relabeling of the x_i, we may assume that there exists an integer $1 \leq a < k$, and real numbers $\lambda_1, \ldots, \lambda_{k-1}$, such that $f(x_k) = \sum_{i=1}^{k} \lambda_i f(x_i)$, $\sum_{i=1}^{k} \lambda_i = 1$, $\lambda_1, \ldots, \lambda_a$ are negative and $\lambda_{a+1}, \ldots \lambda_{k-1}$ are positive.

Then, define $\alpha = \sum_{i=a+1}^{k} \lambda_i$. As the λ_i sum up to 1, all the λ_i are not negative and $\alpha > 0$. Hence, we have:

$$\frac{1}{\alpha} f(x_k) + \sum_{i=1}^{a} \left(-\frac{\lambda_i}{\alpha} \right) f(x_i) = \sum_{i=a+1}^{k-1} \frac{\lambda_i}{\alpha} f(x_i) \tag{3}$$

Thus, $\mathrm{Conv}(f(\{x_1,\dots,x_a,x_k\}))\cap\mathrm{Conv}(f(\{x_{a+1},\dots,x_{k-1}\}))$ is not empty although the faces $\{x_1,\dots,x_a,x_k\}$ and $\{x_{a+1},\dots,x_{k-1}\}$ of Δ (sub-faces of X) are disjoint; we are led to a contradiction. □

Lemma 3. *If* $\mathrm{Conv}(f(X))\cap\mathrm{Conv}(f(Y))$ *is empty for any two disjoint faces* X *and* Y *of an abstract simplicial complex* Δ*, then* $\mathrm{Conv}(f(X))\cap\mathrm{Conv}(f(Y)) = \mathrm{Conv}(f(X)\cap f(Y))$ *holds for any two faces* X *and* Y *of* Δ*.*

Proof. Let X and Y be any two non-disjoint faces of Δ and let π be a point in $\mathrm{Conv}(f(X))\cap\mathrm{Conv}(f(Y))$. The point π may be expressed as a weighted average of the points in $f(X)$: $\pi=\sum_{x\in X}\alpha(x)f(x)$, where α is a mapping from $V(\Delta)$ to $\mathbb{R}^+$ with sum 1 ans support $\mathrm{Supp}(\alpha)\subseteq X$. Similarly, $\pi=\sum_{y\in Y}\beta(y)f(y)$, where β is a function from $V(\Delta)$ to $\mathbb{R}^+$ with sum 1 and support $\mathrm{Supp}(\beta)\subseteq Y$. Let $\lambda(z)=\min(\alpha(z),\beta(z))$. This function is positive and has support $\mathrm{Supp}(\lambda)\subseteq X\cap Y$. Let s be its sum: $s=\sum_{z\in X\cap Y}\lambda(z)$. Let us show that s cannot be different from 1: otherwise,

$$\sum_{x\in X}\left(\frac{\alpha(x)-\lambda(x)}{1-s}\right)f(x)=\sum_{y\in Y}\left(\frac{\beta(y)-\lambda(y)}{1-s}\right)f(y) \tag{4}$$

and, if X' denotes the support of $\alpha-\lambda$ and Y' denotes the support of $\beta-\lambda$, $\mathrm{Conv}(f(X'))\cap\mathrm{Conv}(f(Y'))$ is not empty although the faces $X'\subseteq X$ and $Y'\subseteq Y$ are disjoint. Hence, $s=1$, what may only be achieved by $\alpha=\beta$. Thus, π belongs to $\mathrm{Conv}(f(X)\cap f(Y))$. □

Theorem 2 (Folklore). *Let* Δ *be an abstract simplicial complex and let* $f: V(\Delta)\to\mathbb{R}^n$ *be a mapping. Then,* f *is a geometric realization of* Δ *in* $\mathbb{R}^n$ *if and only if* $\mathrm{Conv}(f(X))$ *and* $\mathrm{Conv}(f(Y))$ *are disjoint, for any two disjoint faces* X,Y *of* Δ*.*

Proof. If f is a geometric realization then disjoint sets in Δ are maped into disjoint simplices. Conversely, if disjoint faces are maped into set having disjoint convex hulls, then the intersection of the convex hulls of the images of any two faces X and Y of Δ is the convex hull of the image of their intersection, according to Lemma 3. Then, according to Lemma 2, f is a geometric realization of Δ in $\mathbb{R}^n$. □

4 Geometric Realization of a d-Representation

In the following, we consider a d-representation $R=(<_1,\dots,<_d)$ of a set V and a mapping f from V to $\mathbb{R}^d$, such that f_i is strictly positive and strictly increasing with respect to $<_i$ (for $1\le i\le d$). We denote by P the image of V and $\Delta(P)$ the image of $\Sigma(R)$. As f is clearly injective, $\Delta(P)$ and $\Sigma(R)$ are isomorphic.

Let $X \subseteq P$ be a subset of points, $\sigma(X)$ denotes the point with coordinates $\sigma_i(X) = \max_{\pi \in X} \pi_i$ and $\Theta(X)$ denotes the closed set $\Theta(X) = \{\pi, \quad \forall 1 \le i \le n, \pi_i \le \sigma_i(X)\}$. According to these definitions and according to the definition of $\Sigma(R)$, a point set $X \subseteq P$ is a face of $\Delta(P)$ if and only if

- any point in X belongs to the frontier of $\Theta(X)$,
- no point of P belongs to the interior of $\Theta(X)$.

Theorem 3. *Let $R = (<_1, \ldots, <_d)$ be a d-representation of a set V. Let f be any mapping from V to $\mathbb{R}^d$, such that f_i is strictly positive and increasing with respect to $<_i$ sufficiently fast (that is: the ratio of consecutive values of f_i is bigger than $A \ge 1 + \sqrt{d}$), let $(\lambda_1, \ldots, \lambda_d)$ be d-uple of positive real numbers different from $(0, \ldots, 0)$ and let H be the hyperplane of $\mathbb{R}^d$ defined by $\sum_i \lambda_i x_i = 1$.*

Then, the mapping $\phi : V \to H$ defined by $\phi_i(x) = \frac{f_i(x)}{\sum_j \lambda_j f_j(x)}$ is a geometric realization of $\Sigma(R)$ in $H \approx \mathbb{R}^{d-1}$.

Proof. We shall first prove that $\Delta(P) * O$ is a geometric simplicial complex, that is that $\mathrm{Conv}(X) \cap \mathrm{Conv}(Y)$ is empty for any two disjoint sets of $\Delta(P) * O$ (according to Theorem 2). As, for any face $X \in \Delta(P) * O$, $X \cup \{O\}$ is also a face of $\Delta(P) * O$, it is necessary and sufficient to prove that $\mathrm{Conv}(X \cup \{O\}) \cap \mathrm{Conv}(Y)$ is empty for any two faces X, Y of $\Delta(P)$ or, equivalently, that there exists an hyperplane H passing through O and which separates X from Y. As no point of X belongs to $\Theta(Y)$, each point $x \in X$ has a coordinate bigger than the corresponding one of $\sigma(Y)$. Hence, the set $I = \{i, \exists x \in X, x_i > \sigma_i(Y)\}$ is not empty. Similarly, the set $J = \{j, \exists y \in Y, y_i > \sigma_i(X)\}$ is also not empty. Let $a(x) = \sum_{i \in I} \frac{x_i}{\sigma_i(Y)}$ and $b(x) = \sum_{j \in J} \frac{x_j}{\sigma_j(X)}$. We have: $a(x) \ge A$ for any $x \in X$ as there exists an index i, for which $x_i > 1$ and hence $x_i \ge A$; To the opposite, $a(y) < 1 + \frac{d-1}{A}$ for $y \in Y$, as only one coordinate of y may reach the maximum value. Similarly, $b(x) < 1 + \frac{d-1}{A}$ for any $x \in X$ and $b(y) \ge A$ for any $y \in Y$. As $A \ge 1 + \sqrt{d}$, we have $A > 1 + \frac{d-1}{A}$. Thus, the hyperplane defined by $a(x) = b(x)$ passes through O and separates X (for which $a(x) > b(x)$ from Y (for which $a(x) < b(x)$) and $\Delta(P) * O$ will be a geometric simplicial complex.

Now, consider the hyperplane H' defined by $\sum_i \lambda_i x_i = \epsilon$, where ϵ is a sufficiently small positive real number, so that H' separates O from P. Then, the intersection of $\Delta(P) * O$ and H' is a geometric realization of $\Delta(P) \approx \Sigma(R)$ and so is its homothetic image defined by the image of $\Sigma(R)$ by the mapping ϕ. □

5 Triangulation

Lemma 4. *Let $R = (<_1, \ldots, <_d)$ be a d-representation of V with complex $\Sigma(R)$ and let x be the maximum of $<_k$. For $i \ne k$, let $<'_i$ be the total order on V where x precedes all the elements of $V \setminus \{x\}$ and the element of $V \setminus \{x\}$ are ordered by $<_i$. Then $R' = (<'_1, \ldots, <'_{k-1}, <_k, <'_{k+1}, \ldots, <_d)$ is a d-representation which complex $\Sigma(R')$ includes $\Sigma(R)$.*

Proof. Denote by S' the supremum section corresponding to R'. Let X be a subset of V, then $S(X) \subseteq S'(X)$:

- If $x \notin X$, $\quad S'(X) = S(X)$,
- If $x \in X$, $\quad S'(X) = \{x\} \cup S(X \setminus \{x\}) \supseteq S(X)$

Thus, any element X of $\Sigma(R)$ belongs to $\Sigma(R')$: for all $v \in V, v \in S(X \cup \{v\}) \subseteq S(X \cup \{v\}$ and hence $X \in \Sigma(R')$. ☐

A d-representation $R = (<_1, \ldots, <_d)$ on a set V is *standard* if $|V| \geq d$ and, for all $i \neq j$, the maximum element of $<_i$ is one of the $d-1$ smallest elements of $<_j$. The maxima of the orders of a standard d-representation are the *exterior* elements of this representation. The other elements of V are the *interior* elements. An abstract simplicial complex is *standard* if it is the complex of some $\dim P(\Delta)$-representation. Notice that, for a standard abstract simplicial complex Δ, we have: $\dim \Delta = \dim P(\Delta) - 1$.

Given any subset X of V and a d-representation R of V, we define the *shade function* I_X of X on $V \setminus X$ as $I_X(u) = \{i \in [1, d], \forall x \in X, u >_i x\}$. Given any subset X of V and a d-representation R of V, we define the *shading order* $<_X$ of X on $V \setminus X$ as follows: $u <_X v \iff (I_X(u) \subseteq I_X(v))$ and $(\forall i \in I_X(u), u <_i v)$

Lemma 5. *Any standard representation will be pure. More precisely, let* $R = (<_1, \ldots, <_d)$ *be a standard representation. If* X *is a face of* $\Sigma(R)$ *which maximal element* x_1 *(with respect to* $<_1$*) is not an exterior element of* R*, then, for every* $\dim X < k < d$ *there exists a* k*-dimensional face which includes* X *and has* x_1 *as a maximum element with respect to* $<_1$.

Proof. Let X be an element of $\Sigma(R)$ different from the $|X|$ first elements of $<_1$ and assume $|X| < d$.

If X is included into the set $\{v_1, \ldots, v_k\}$ of the external elements of R, the addition of any external element smaller or equal to x_1 with respect to $<_1$ will do.

Otherwise, let x be an element of x which is maximal in X with respect to two different total orders and let $<_k$ be one of them for which $k \neq 1$. Obviously, x is not an external element of R as an external element is greater than an internal with respect to exactly one total order. For the same reason, v_k does not belong to X and is smaller or equal to x_1 with respect to $<_1$. Let α be a minimal element of the set $\{v \notin X, v <_1 x_1 \text{ and } v <_X v_k\} \cup \{v_k\}$. As $I_X(\alpha) \neq \emptyset$ (otherwise, $\alpha \notin S(X+\alpha)$ would contradict $X \in \Sigma(R)$) and as $I_X(\alpha) \subseteq I_X(v_k)$ by construction, we get: $I_X(\alpha) = \{k\}$. This ensures that $S(X+\alpha) = X+\alpha$. Assume there exists an element $y \notin X + \alpha$, such that $y \notin S(X + \alpha + y)$. As $X \in \Sigma(R)$, the element y belongs to $S(X + y)$. As $I_X(\alpha) = \{k\}$, the only possibility for y not to belong to $S(X + \alpha + y)$ corresponds to the situation where $I_X(y) = \{k\}$ and $y <_k \alpha$, that is: $y <_X \alpha$. Moreover, y is smaller or equal to x_1 with respect to $<_1$ (as $1 \notin I_X(y)$) and this contradicts the minimality of α. ☐

Theorem 4. *Any abstract simplicial* Δ *complex may be triangulated into a standard representation having the same face poset dimension.*

Proof. As a direct consequence of Theorem 1, Δ is the subcomplex of some d-representation R_0. By successive applications of Lemma 4, there exists a standard d-representation R, such that $\Sigma(R)$ includes $\Sigma(R_0)$ and hence includes Δ. According to Lemma 5, $\Sigma(R)$ is pure of dimension $d-1$ and hence is a triangulation of Δ. $\square$

6 Shellability of Standard Complexes

Shellability of pure simplicial complexes has been extensively studied. We shall introduce here the *vertex shellability*, which is a generalization of the concept introduced in [8] and which has proved its efficiency through numerous applications in quite different kind of problems related to planarity (see, for instance, [6][7][9][13]). We shall prove that the concepts of shellability and vertex-shellability are actually equivalent. Thatfor, recall that a pure abstract simplicial complex Δ of dimension $d-1$ is *shellable* if all its $(d-1)$-faces (that is: all its elements of cardinality d) can be listed $F_1, \ldots, F_s$ in such a way that $\left(\bigcup_{i=1}^{j-1} \overline{F_i}\right) \cap \overline{F_j}$ is pure of dimension $d-2$ for every $1 < j \leq s$ (where $\overline{F_i}$ is defined by: $\overline{F_i} = \{X \in \Delta, X \subseteq F_i\}$) or, equivalently, for $1 \leq i < j \leq s$, any subset X of F_i and F_j, there exists a $(d-2)$-dimensional face $Y \supseteq X$ and a $(d-1)$-dimensional face F_h (for some $h < j$) such that Y is included in both F_h and F_j.

A pure abstract simplicial complex Δ of dimension $d-1$ is said to be *vertex shellable* if all its vertices can be listed $v_1, \ldots, v_n$ in such a way that:

- $\{v_1, \ldots, v_d\} \in \Delta$,
- for all face $\sigma \in \Delta$ with maximum vertex v_k $(k > d)$, there exists $j < k$, such that $v_j \notin \sigma$ and $\sigma - v_k + v_j \in \Delta$),
- for all vertex v_k $(k \geq d)$, the abstract simplicial complex $\Delta_k = \{\sigma - v_k, \sigma \in \Delta \text{ and } \sigma \subseteq \{v_1, \ldots, v_k\}\}$ is a pure $d-2$ dimensional shellable simplicial complex.

Lemma 6. *Let Δ be a pure $d-1$ dimensional shellable simplicial complex and let $F_1, \ldots, F_s$ be a shelling order.*

Assume there exists $d < a < s$, such that F_a is not included in $\bigcup_{i<a} F_i$ but F_{a+1} is. Then, $F_1, \ldots, F_{a-1}, F_{a+1}, F_a, F_{a+2}, \ldots, F_s$ is also a shelling order.

Proof. As $\left(\bigcup_{i<a} \overline{F_i}\right) \cap \overline{F_a}$ is a pure $d-2$ dimensional simplicial complex and as there exists $\alpha \in F_a \backslash \bigcup_{i<a} F_i$, we get that there exists $i < a$, such that $F_a - \alpha \subset F_i$. As $\left(\bigcup_{i<a} F_i\right) \cup F_{a+1}$ does not include α, we get that $\left(\bigcup_{i<a} \overline{F_i} \cup \overline{F_{a+1}}\right) \cap \overline{F_a}$ is a pure $d-2$ simplicial complex.

Moreover, let X be a face of both F_{a+1} and F_j $(j < a)$, with dimension strictly less than $d-2$. Then, as $F_1, \ldots, F_s$ is a shelling order, there exists a $(d-2)$-dimensional face $Y \supseteq X$ which is a face of F_{a+1} and F_i (with $i < a+1$). If $i = a$ then, as $Y \subset F_{a+1}$, the vertex α does not belong to Y and $Y = F_a - \alpha$. Hence, there exists $j < a$ such that $Y \subset F_j$. Altogether, $\left(\bigcup_{i<a} \overline{F_i}\right) \cap \overline{F_{a+1}}$ is a pure $(d-2)$-dimensional complex. $\square$

Lemma 7. *Let Δ be a pure $(d-1)$-dimensional abstract simplicial complex. If Δ is shellable, then it is vertex shellable.*

Proof. Assume Δ is shellable. According to Lemma 6, there exists a shelling order $F_1, \ldots, F_s$ such that, for all $d < a < s$, if F_a is not included in $\bigcup_{i<a} F_i$, no F_b (with $b > a$) is included in $\bigcup_{i<a} F_i$. Hence, as $F_i \setminus \left(\bigcup_{j<i} F_j\right)$ includes at most one element, we can list the vertices of Δ the following way:

- begin with a list including the vertices of F_1 and let $a = 1$.
- while $a < s$, add to the list the vertex (if any) of F_{a+1} which is not in the list and let $a \leftarrow a + 1$.

Eventually, we get a list $v_1, \ldots, v_n$ of all the vertices of Δ, such that if $i < j$, all the $(d-1)$-faces of Δ with maximum element v_i precedes all the $(d-1)$-faces including v_j in the shelling order. Thus, we get:

- $\{v_1, \ldots, v_d\} \in \Delta$.
- for all face $\sigma \in \Delta$ with maximum vertex v_k $(k > d)$, there exists a face σ' that precedes σ in the shelling order, such that $\sigma - v_k \in \sigma'$. As σ' precedes σ and does not include v_k, its maximal element is a vertex v_i with $i < k$. Hence, the vertex v_j such that $\sigma' = \sigma - v_k + v_j$ is such that $j < k$.
- for all vertex v_k $(k \geq d)$, let Δ_k be the abstract simplicial complex $\Delta_k = \{\sigma - v_k, \quad \sigma \in \Delta \text{ and } \sigma \subseteq \{v_1, \ldots, v_k\}\}$. Let F_a and F_b be the first and last $(d-1)$-dimensional face of the shelling order having v_k as maximal element. We shall prove that Δ_k is a pure $(d-2)$-dimensional abstract simplicial complex having $F_a - v_k, \ldots, F_b - v_k$ as a shelling order: Consider in Δ_k a face X of $F_i - v_k$ and $F_j - v_k$ with $a \leq i < j \leq b$. Then, $X + v_k$ is a face of F_i and F_j in Δ. Thus there exists a $(d-2)$-dimensional face $Y \supseteq X + v_k$ and $h < j$, such that $Y \subseteq F_h$. As v_k belongs to F_h, we get $h \geq a$. Hence, there exists a face $F_h - v_k$ of Δ_k having a $(d-3)$-dimensional face $Y - v_k$ including X and included in F_j, what ends our proof.

$\square$

Theorem 5. *Let Δ be a pure $(d-1)$-dimensional abstract simplicial complex. Then, Δ is shellable if and only if it is vertex shellable.*

Proof. According to Lemma 7, we only have to prove that the vertex shellability of Δ will imply its shellability.

Let $v_1, \ldots, v_n$ be a vertex shelling order. We list the $(d-1)$-faces of Δ the following way:

- let $F_1 = \{v_1, \ldots, v_d\}$ and let $a = d$.
- while $a < s$ we add to the list the faces of Δ having v_k as a maximal element in the order induced by the shelling order of Δ_k.

Let $1 \leq i < j \leq s$ and let X be a subset of F_i and F_j and let v_k be the maximal element of F_j.

– If $v_k \notin X$, then there exists a $(d-1)$-face F_h including $F_j - v_k$ with $h < j$, according to the definition of a vertex shelling order. Obviously, $X \subseteq F_j - v_k$.
– If $v_k \in X$, then there exists a $(d-3)$-face $Y \supseteq X - v_k$ of Δ_k and a face F_h having v_k as a maximal element, such that $h < j$ and $F_h - v_k \supseteq Y$, according to the shelling order of Δ_k. Hence, $Y + v_k$ is a $(d-2)$-face included in both F_h and F_j (with $h < j$).

□

Let $R = (<_1, \ldots, <_d)$ be a d-representation. Let X and Y be two $(d-1)$-dimensional simplices of $\Sigma(R)$ and let x_i (resp. y_i) be the maximal element of X (resp. Y) with respect to $<_i$ (notice that $X = \{x_1, \ldots, x_d\}$ and $Y = \{y_1, \ldots, y_d\}$), the *d-order* is the total order on the $(d-1)$-dimensional simplexes of $\Sigma(R)$ which is defined by:

$$\forall X \neq Y \in \Sigma(R), \quad (X < Y \iff x_k <_k y_k, \text{ where } k = \min\{i, x_i \neq y_i\}) \quad (5)$$

Theorem 6. *Any standard abstract simplicial complex will be shellable.*

More precisely, let $R = (<_1, \ldots, <_d)$ be a standard representation. Then, the d-order is a shelling order of $\Sigma(R)$ and any of the $<_i$ is a vertex-shelling order.

Proof. We shall only prove that the d-order is a shelling order, as this obviously implies that $<_1$ is a vertex shelling order (according to the definitions of a vertex shelling order and the fact that the d-order is a lexicographic order starting with $<_1$) and hence that $<_i$ is a vertex shelling order (by symmetry).

We shall prove that the d-order is a shelling order by induction over the dimension d of the simplicial complex.

Let F_i be the ith face $(i > 1)$ and let x be its maximal element with respect to $<_1$. As F_1 is the only face including the $d-1$ exterior elements $v_2, \ldots, v_k$, there exists, according to Lemma 5 a maximal face F_j including $F_i - x$ and which maximal element (with respect to $<_1$) is strictly smaller than x and hence $j < i$. Let F_a $(a \leq i)$ be the first simplicial complex containing x, $\left(\bigcup_{j=1}^{a-1} \overline{F_j}\right) \cap \overline{F_i} = \overline{F_i - x}$ and, according to the induction on the dimension, if $a < i$, $\left(\bigcup_{j=a}^{i-1} \overline{F_j - x}\right) \cap \overline{F_i - x}$ is a pure simplicial complex of dimension $d-3$ and hence, $\left(\bigcup_{j=a}^{i-1} \overline{F_j}\right) \cap \overline{F_i}$ is a pure simplicial complex of dimension $d-2$. Altogether, $\left(\bigcup_{j=1}^{i-1} \overline{F_j}\right) \cap \overline{F_i}$ is a pure simplicial complex of dimension $d-2$. □

Theorem 7. *Let Δ be an abstract simplicial complex and $d = \dim P(\Delta)$. Then, there exists a standard d-representation $R = (<_1, \ldots, <_d)$ $V(\Delta)$ defining a triangulation Δ^+ of Δ, which is shellable and realizable in $\mathbb{R}^{d-1}$.*

Proof. According to Theorem 4, Δ may be triangulated into $\Delta^+ = \Sigma(R)$, where R is a standard d-representation $V(\Delta)$. Δ^+ is shellable, according to Theorem 6 and its geometric realization follows from Theorem 3, using any sufficiently fast increasing functions. □

Acknowledgments

The author would like to thank Jiri Matousek for his helpfull comments and remarks.

References

1. L. Barany, R. Howe, and H. Scarf, *The complex of maximal lattice free simplices*, Mathematical Programming **66 (Ser. A)** (1994), 273–281.
2. D. Bayer, I. Peeva, and B. Sturmfels, *Monomial resolutions*, AMS electronic preprint #199610-14-012, 1996, (a.k.a. alg-geom/9610012).
3. G. Brightwell and W.T. Trotter, *The order dimension of convex polytopes*, SIAM Journal of Discrete Mathematics **6** (1993), 230–245.
4. ———, *The order dimension of planar maps*, SIAM jouranl on Discrete Mathematics **10** (1997), no. 4, 515–528.
5. H. de Fraysseix and P. Ossona de Mendez, *On topological aspects of orientations*, Proc. of the Fifth Czech-Slovak Symposium on Combinatorics, Graph Theory, Algorithms and Applications, Discrete Math., (to appear).
6. ———, *Regular orientations, arboricity and augmentation*, DIMACS International Workshop, Graph Drawing 94, Lecture notes in Computer Science, vol. 894, 1995, pp. 111–118.
7. H. de Fraysseix, P. Ossona de Mendez, and P. Rosenstiehl, *On triangle contact graphs*, Combinatorics, Probability and Computing **3** (1994), 233–246.
8. H. de Fraysseix, J. Pach, and R. Pollack, *Small sets supporting Fary embeddings of planar graphs*, Twentieth Annual ACM Symposium on Theory of Computing, 1988, pp. 426–433.
9. ———, *How to draw a planar graph on a grid*, Combinatorica **10** (1990), 41–51.
10. B. Dushnik, *Concerning a certain set of arrangements*, Proceedings of the AMS, vol. 1, 1950, pp. 788–796.
11. B. Dushnik and E.W. Miller, *Partially ordered sets*, Amer. J. Math. **63** (1941), 600–610.
12. T. Hibi, *Algebraic combinatorics on convex polytopes*, Carlslaw Publications, 1992.
13. G. Kant, *Algorithms for drawing planar graphs*, Ph.D. thesis, Utrecht University, Utrecht, 1993.
14. H. Scarf, *The computation of economic equilibria*, Cowles foundation monograph, vol. 24, Yale University Press, 1973.
15. W. Schnyder, *Planar graphs and poset dimension*, Order **5** (1989), 323–343.

Visibility Representations of Complete Graphs

Robert Babilon, Helena Nyklová*, Ondřej Pangrác, and Jan Vondrák

Department of Applied Mathematics
Charles University
Malostranské nám. 25, Prague, Czech Republic
{babilon,nyklova,pangrac,vondrak}@kam.ms.mff.cuni.cz

Abstract. In this paper we study 3-dimensional visibility representations of complete graphs. The vertices are represented by equal regular polygons lying in planes parallel to the xy-plane. Two vertices are adjacent if and only if the two corresponding polygons see each other - i.e. it is possible to construct an abscissa perpendicular to the xy-plane connecting the two polygons and avoiding all the others.
We give the bounds for the maximal size $f(k)$ of a clique represented by regular k-gons: $\lfloor \frac{k+1}{2} \rfloor + 2 \leq f(k) \leq 2^{2^k}$ and we present a particular result for triangles: $f(3) \geq 14$.

1 Introduction

Consider a finite number of equal regular k-gons located in parallel planes so that their corresponding edges are parallel (we can only shift each polygon in its plane). We say that two polygons can see each other if there is a line segment perpendicular to the planes, which connects the two polygons and does not intersect any other. Let $f(k)$ denote the maximum number of k-gons which can be placed so that each pair of them can see each other.

Recently the result is known for squares (maximum clique represented by squares is K_7) and for discs (a clique of arbitrary size can be represented by discs). Both these result were shown by Fekete, Houle and Whitesides in [FHW].

We can assume that the planes are perpendicular to the z-axis, and we consider the projection of the polygons into the xy-plane. Clearly, only the z-order of the polygons is important, not the exact z-coordinates. We number the polygons according to the ordering on z-coordinate. We encode the position of each polygon by k coordinates (which are not independent); each of them is measured in the direction perpendicular to one of the edges. (For example, we have 3 coordinates for a triangle, but only two of them are independent.) The basic idea is that the ordering of the polygons in these coordinates captures the geometric properties of our configuration completely. Note that although the exact values of two coordinates determine the location of the triangle uniquely, the ordering in two coordinates is not sufficient to describe the configuration. In Figure 1, the ordering in coordinates π and ρ is the same, but the σ-ordering of the triangles differ and the two configurations are not equivalent.

* Research supported by grant GAUK 159/99

J. Kratochvíl (Ed.): GD'99, LNCS 1731, pp. 333–341, 1999.

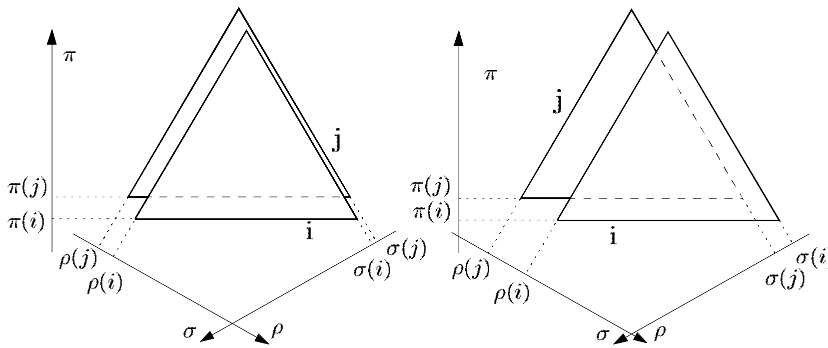


Fig. 1.

Without loss of generality we can assume that no two coordinates are equal, thus for any pair $i \neq j$ of k-gons we have $\pi(i) \neq \pi(j)$ on any axes.

From now on we use the expression *configuration* for a system of k-gons such that every k-gon sees all the others.

2 General properties

Lemma 1 *If we allow rotation (i.e. we do not insist on parallelism of the corresponding edges) we can construct the arbitrarily large configuration of half-planes (and k-gons for arbitrary k).*

Proof. Consider a point A and a half-plane ρ such that A lies in the interior of ρ. First construct a half-plane ρ_1. Rotate it by the angle $\alpha_1 = 45^\circ$ with the center in A and we obtain half-plane ρ_2. Every other half-plane we obtain as a rotated copy of the previously constructed half-plane, where $\alpha_i = \alpha_{i-1}/2$. And we have a configuration of arbitrary number of half-planes. See Figure 2. The bounding line of ρ_i is crossing the boundaries of $\rho_1, \rho_2, \ldots, \rho_{i-1}$ and therefore ρ_i sees ρ_j, $j < i$. □

Claim 2 *For every natural k, $f(k) \geq \left\lfloor \frac{k+1}{2} \right\rfloor + 2$.*

Proof. Consider the system of $\left\lfloor \frac{k+1}{2} \right\rfloor$ half-planes seeing each other such that their angles with the horizontal x-coordinate are $0, \frac{2\pi}{k}, 2\frac{2\pi}{k}, 3\frac{2\pi}{k}, \ldots, \left\lfloor \frac{k-1}{2} \right\rfloor \frac{2\pi}{k}$. This can be done by taking the horizontal line p_1 with an orientation from the left to the right side and choosing a point A not lying on p_1 as a center point of the rotation. Every next line p_{i+1} arises from the previous p_i through the counterclockwise rotation by the angle $\frac{2\pi}{k}$ and shifting by δ along the z-coordinate. The half-planes are determined by the lines p_i and the point A which lies on them.

Now take sufficiently large k-gons P_i (so that every P_i contains all the intersection points of p_i and all the other p_j) and align them to the boundaries of the corresponding half-planes. In such a way we obtain $\left\lfloor \frac{k+1}{2} \right\rfloor$ k-gons seeing each other.

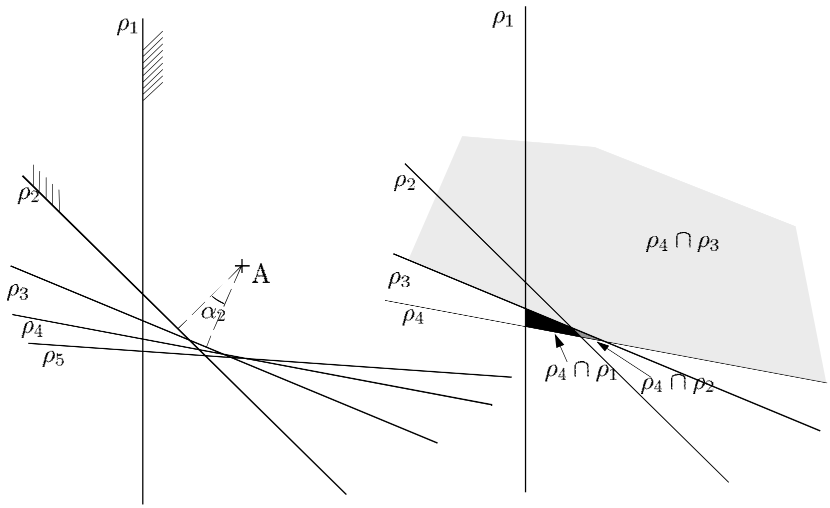


Fig. 2. Half-planes, if we allow rotation.

Moreover, we can add two more k-gons, one to the top and the second to the bottom of the configuration, centers of them lying not far from the intersection point of the first (p_1) and the last ($p_{\lfloor \frac{k+1}{2} \rfloor}$) line. These new k-gons see each other (because the intersection of complements of the first and the last half-plane is not empty) and also see all the reminding k-gons.

There are together (for each k) at least $\lfloor \frac{k+1}{2} \rfloor + 2$ k-gons seeing each other.

The example for the case $k = 6$ is shown in Figure 3. We have three half-planes (lines), three hexagons adjacent to the respective lines (2, 3 and 4), one hexagon on the top (1) and one hexagon on the bottom (5). There are together $\lfloor \frac{6+1}{2} \rfloor + 2 = 5$ hexagons. □

Corollary 3

$$\lim_{k \to \infty} f(k) = \infty.$$

Proof. Immediately follows from the lower bound of $f(k)$.

Furthermore if we take the limit as $k \to \infty$ the k-gon approaches the shape of a disc. For discs we can use the same construction as we used in the proof of Lemma 1. The boundary lines of the half-planes are the touching lines of the corresponding discs and $|\rho_i, A| < r$. We have the same result as shown in [FHW] - every complete graph can be represented by discs. □

Claim 4 *For every natural k, $f(k) \leq 2^{2^k}$.*

Lemma 5 *If there is a sequence of three polygons which is monotone in each ordering, then there are two polygons such that one of them does not see the other one.*

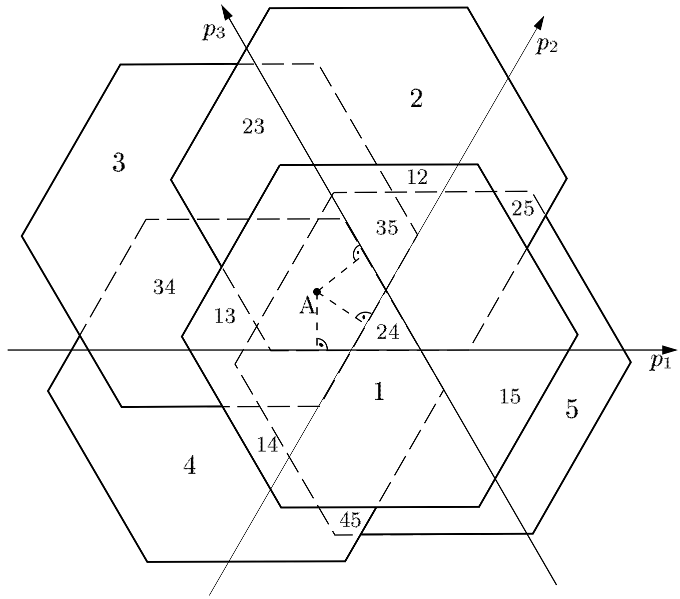


Fig. 3. $f(6) \geq 5$

Proof. If we have a sequence of three polygons monotone in each ordering, the middle one (in z-order) is the middle one in the ordering on each coordinate. Hence the common area of the first and the third one is contained in the second one. So they cannot see each other. (See Fig. 4.) □

Proof of Claim 4. We use the Erdős-Szekeres theorem, which says that any sequence of $(a-1)(a-1)+1$ distinct numbers contains a monotone subsequence of the length a. We have a sequence $\{(x_1, x_2, \dots, x_k)_i\}_{i=1}^{n} \in \mathbb{N}^k$ and we want to find a subsequence of the length 3 monotone in each coordinate. We proceed by induction on k.

Let us have a sequence $\{x_i\}_{i=1}^{n}$, $x_i \in \mathbb{N}$. Hence by Erdős-Szekeres theorem if $n_1 = (3-1)(3-1)+1 = 5 = 2^{2^1}+1$ for $k = 1$ we can find a monotone subsequence of the length 3. For $n_k = (n_{k-1}-1)(n_{k-1}-1)+1 = \left(2^{2^{k-1}}\right)\left(2^{2^{k-1}}\right)+1 = 2^{2^k}+1$ there is a subsequence of the length $\left(2^{2^{k-1}}+1\right)$ monotone in the first coordinate. By induction this subsequence contains a 3-element subsequence monotone in all coordinates except the first one. But the whole subsequence is monotone in the first coordinate. So we have 3-element subsequence monotone in each coordinate. Hence by the previous lemma there are two k-gons such that one of them does not see the other one. □

If k is even we have a better estimate, as it is necessary to use only $k/2$ coordinates. Hence we can get $f(2k) \leq 2^{2^k}$. The same proof can be used to prove the theorem for polygons of different sizes with parallel edges.

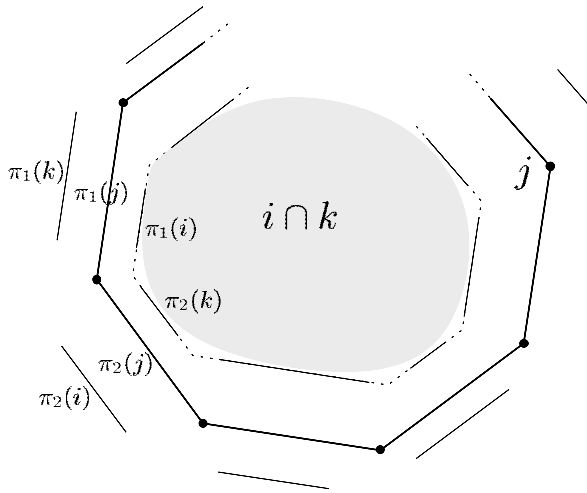


Fig. 4. Sequence of three polygons $i < j < k$ monotone in each coordinate

Conjecture 1 *For every natural $k > 4$, we conjecture $f(k) \leq f(k+2)$.*

Conjecture 2 *For every natural $k > 4$, we conjecture $f(k) \leq f(2k)$.*

Conjecture 3 *For every even k it is possible to draw up an algorithm, which tells the maximum size of a configuration of k-gons, if we additionally require that the intersection of every pair of k-gons is a k-gon again.*

The algorithm is similar to the one used in [FHW] for enumerating the maximum size of configuration of squares. If the k-gons lying in between i and j cover all k vertices of the intersection $i \cap j$, then i cannot see j. Thus all k-gons in the configuration see the others if and only if the following holds for every pair $i < j$:

$$\left| \bigcup_{i<m<j} \bigcap_{1 \leq l \leq k} P_l(m) \right| < k,$$

where $P_l(m)$ is the set of vertices of the intersection $i \cap j$, which can be covered by m in the direction of the π_l-coordinate.

3 Triangles

Claim 6

$$f(3) \geq 14.$$

Proof. An example of configuration of fourteen triangles, which see each other is given in Figure 6. The orderings on the π, ρ and σ coordinates are given in the following table.

π	5	4	10	8	7	2	6	1	3	14	13	12	11	9
ρ	9	4	13	11	14	12	7	3	1	6	2	10	8	5
σ	12	14	9	11	5	1	13	8	10	2	3	6	7	4

□

We have developed a functional implementation of an algorithm which enumerates the upper bound for $f(3)$. It is based on the following three conditions the orderings on the three coordinates π, ρ a σ must satisfy, otherwise there is a pair of triangles not seeing each other.

Lemma 7 *Every configuration of k-gons, has the following two properties:*

1 *There is no pair of k-gons $i \neq j$ such that*

$$\forall n = 1, .., k : \pi_n(i) < \pi_n(j),$$

2 *there is no triple $i < j < l$ such that*

$$\forall n = 1, .., k : ((\pi_n(i) < \pi_n(j) < \pi_n(l)) \vee (\pi_n(i) > \pi_n(j) > \pi_n(l)) .$$

Remark: For even k we use only $\pi_1, \ldots, \pi_{k/2}$- axis to describe the configuration; configuration satisfies **1** *automatically.*

First property follows from the fact that we have equal k-gons and therefore no k-gon can be smaller that any other in every coordinate. The second property is only a re-formulation of Lemma 5, which says that there cannot be a subsequence monotone in all of the three coordinates, otherwise there is a pair of triangles not seeing each other. □

Lemma 8 *Every configuration of triangles has the following property:*

3 *If there is a subsequence monotone in the two coordinates, then the middle triangle must have the highest order in the third one.*

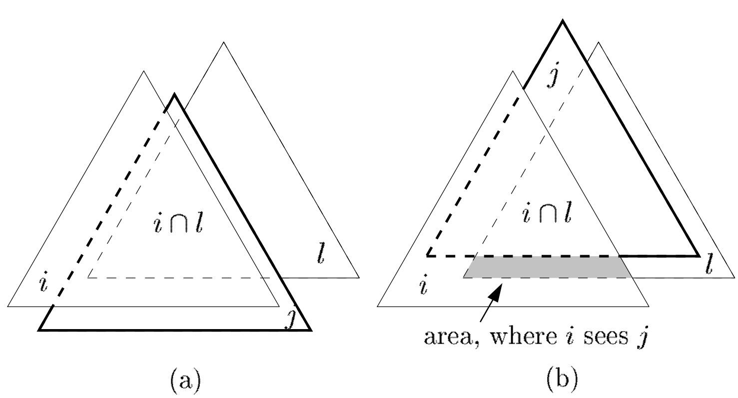


Fig. 5. Illustration to the property **3**.

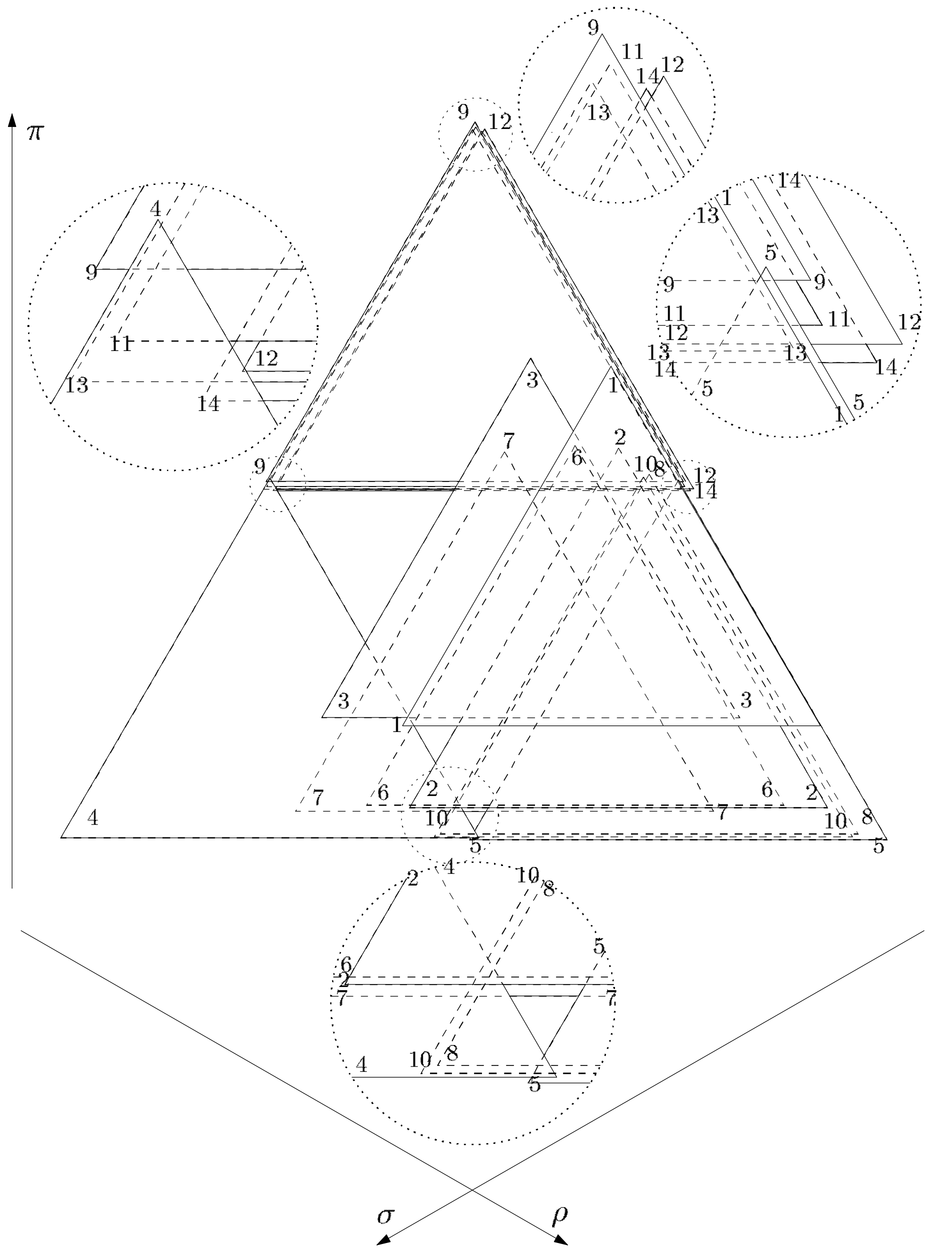


Fig. 6. Fourteen triangles seeing each other.

Thus there is no triple $i < j < l$ such that

$$((\pi(i) < \pi(j) < \pi(l)) \vee (\pi(i) > \pi(j) > \pi(l)))$$
$$\wedge ((\rho(i) < \rho(j) < \rho(l)) \vee (\rho(i) > \rho(j) > \rho(l)))$$
$$\wedge ((\sigma(j) < \sigma(i) < \sigma(l)) \vee (\sigma(j) < \sigma(l) < \sigma(i)))\,.$$

Proof. See Figure 5. We have three triangles $i < j < l$. In both figures is $(\rho(i) < \rho(j) < \rho(l))$ and $(\sigma(l) < \sigma(j) < \sigma(i))$ - a subsequence monotone in the two coordinates. In Figure 5(a) is $(\pi(j) < \pi(i) < \pi(l))$ and j covers the intersection $i \cap j$ entirely, but in Figure 5(b), where $(\pi(i) < \pi(l) < \pi(j))$, there is an area, where i can see j. □

We have stated two conjectures in connection with triangles.

Conjecture 4 *Configuration is realizable (with no regard to the visibility) if and only if the orderings satisfy the property* **1**.

Conjecture 5 *It is possible to construct a configuration of triangles seeing each other if and only if the three orderings satisfy the properties* **1**, **2** *and* **3**.

The result we obtain from the computer is a triple of sequences satisfying the necessary conditions - the upper bound. We have to give an example of the corresponding number of triangles each seeing all the others. We would not need that once we have proved Conjecture 5.

After searching through approximately one third of all possibilities, we have the lower bound $f(3) \leq 14$ and we conjecture that it is the final result.

4 Open problems

1 *To prove or refute the conjectures mentioned in the previous.*

2 *We want to draw all combinatorially distinct configurations of triangles into a triangle net. How fine this net has to be ?*

3 *What can be said about the function $f(k)/k$? Does the limit* $\lim_{k\to\infty} \frac{f(k)}{k}$ *exist ?*

References

AGW. H. Alt, M. Godau, S. Whitesides: Universal 3-Dimensional Visibility Representations for Graphs *Proc. Graph Drawing '95*, Passau, 1995. Lecture Notes in Computer Science LNCS #1027, Springer-Verlag, 1996, pp. 8-19

FHW. S. P. Fekete, M. E. Houle, S. Whitesides: New Results on a Visibility Representation of Graphs in 3D *Proc. Graph Drawing '95*, Passau, 1995. Lecture Notes in Computer Science LNCS #1027, Springer-Verlag, 1996, pp. 234-241

Triangle-Free Planar Graphs as Segments Intersection Graphs

N. de Castro[1], F. J. Cobos[1], J.C. Dana[1], A. Márquez[1], and M. Noy[2]

[1] Departamento de Matemática Aplicada I
Universidad de Sevilla, Spain
{natalia,cobos,dana,almar}@cica.es
[2] Dpto. de Matemática Aplicada II
Universitat Politècnica de Catalunya***, Spain
noy@grec.upc.es

Abstract. We prove that every triangle-free planar graph is the graph of intersection of a set of segments in the plane. Moreover, the segments can be chosen in only three directions (horizontal, vertical and oblique) and in such a way that no two segments cross, i.e., intersect in a common interior point.

1 Introduction

Given a set S of segments in the plane, its *intersection graph* has a vertex for every segment and two vertices are adjacent if the corresponding segments intersect. Intersection graphs of segments and other geometrical objects have been widely studied in the past.

For instance, if the segments are contained in a straight line then we have the *interval graphs* [4], a well-known family of perfect graphs. If the segments are chords of a circle then the intersection graph is called a *circle graph*, see for instance [6,8].

In the general case, there is no satisfactory characterization, but some results are known for planar graphs. The most interesting one is due to de Fraysseix, Osona de Mendez and Pach [3], and independently to Ben-Arroyo Hartman, Newman and Ziv [2], which says that every planar bipartite graph is the intersection graph of a set of horizontal and vertical segments (on the other hand, it is known that the recognition of such graphs is an NP-complete problem [5]).

This result provides a partial answer to a question of Scheinerman [7]: *is every planar graph the intersection graph of a set of segments in the plane?*

The main result in this paper, which is a significant extension of [3], is that every triangle-free planar graph is the intersection graph of a family of segments. Moreover, the segments can be drawn in only three directions and in such a way that they do not cross. This particular class of intersection graphs is also known as *contact* graphs. We call such a representation *a segment representation* of the graph.

*** Partially supported by *DGES-MEC-PB96-0005-C02*

J. Kratochvíl (Ed.): GD'99, LNCS 1731, pp. 341–350, 1999.

A key point in our proof is Grötszch's Theorem [9], which guarantees that every planar triangle-free graph is 3-colorable. The sketch of the proof is as follows. Given a triangle-free plane graph G, adding new vertices and (induced) paths between the vertices of G we can obtain a new triangle-free plane graph which is a subdivision of a 3-connected graph. Starting with a 3-coloring, a segment representation in three directions is obtained for this new graph using several technical lemmas. Finally, removing the dummy vertices and paths, we obtain a segment representation of the graph G. The three directions considered are horizontal, vertical and oblique (parallel to the bisector of the second quadrant of the plane).

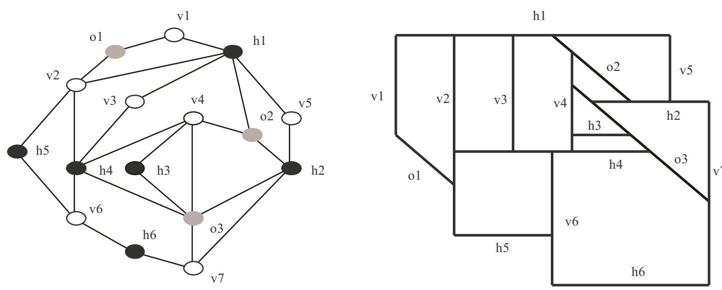


Fig. 1. A segment representation of a planar graph.

2 Convex Faces with Three Directions

Let I_G be a segment representation of a plane graph G. A *segment path* of length n is a sequence of segments $P = \{s_1, \ldots, s_n\}$ such that segment s_i is adjacent to s_{i-1} and s_{i+1} for $1 < i < n$. The representation I_G divides the plane into regions that we call *faces* of the representation and which correspond to the faces of G. A face F of a segment representation is *convex* if its boundary can be divided into four paths (clockwise) P_1, P_2, P_3 and P_4 such that if s is a segment with extremes in the points (x_1, y_1) and (x_2, y_2) then:

1. if s is in P_1, then $x_1 < x_2$ if s is horizontal, $y_1 > y_2$ if s is vertical and $x_1 < x_2$ and $y_1 > y_2$ if s is oblique;
2. if s is in P_2, then $x_1 > x_2$ if s is horizontal, $y_1 > y_2$ if s is vertical and $x_1 > x_2$ and $y_1 < y_2$ if s is oblique;
3. if s is in P_3, then $x_1 > x_2$ if s is horizontal, $y_1 < y_2$ if s is vertical and $x_1 > x_2$ and $y_1 < y_2$ if s is oblique;
4. if s is in P_4, then $x_1 < x_2$ if s is horizontal, $y_1 < y_2$ if s is vertical and $x_1 < x_2$ and $y_1 > y_2$ if s is oblique.

It can be seen that the partition of a convex face into P_1, P_2, P_3 and P_4 is not unique, for instance in the example of Figure 2 the first segment of P_1 could be in P_4. If the segments $s_1, \dots, s_n$ are the intersection between two convex faces F and G, we will assign those segments to only one path in each face, following the newt rule: if this segments belongs to P_1 in F, then they belongs to P_3 in G and reciprocally. Analogously if they belongs to P_2 in F, then they are in P_4 of G and reciprocally. This rule alow us to fix the partition of a convex face into the four paths depending on the adjacent faces if the partition is not unique.

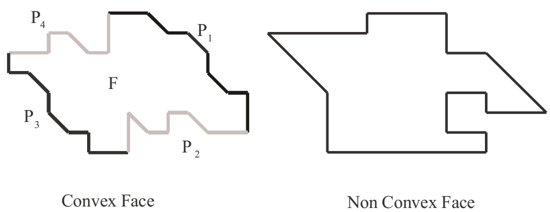


Fig. 2. The four paths of a convex face.

Lemma 1. *Let I_G be a segment representation of a plane graph G such that all its faces are convex, let k be a positive real number and let s be a segment of length l in a face F of I_G. Then, I_G can be transformed into another segment representation I'_G satisfying:*

1. *I_G and I'_G have the same faces and face boundaries as G, and the faces of I'_G are convex;*
2. *the length of s in I'_G is $k + l$;*
3. *at most two segments of F (other than s) have different length in I_G and I'_G.*

Proof. Let P_1, P_2, P_3 and P_4 denote the paths which make F convex. Call the *upper part* (the *lower part*) of the face the union of the paths P_4 and P_1 (P_2 and P_3, respectively). Similarly, the *right part* (the *left part*) is the union of the paths P_1 and P_2 (P_3 and P_4, respectively).

The proof is by induction on the number of faces of the graph G. Suppose that I_G is just the cycle F and let s be the segment of F we want to enlarge. Without loss of generality we can suppose that s is a horizontal segment.

If s is in the upper part of F, we seek another horizontal segment s' in the lower part of F. If there exists such a segment, we enlarge s and s' by the same amount and we make a translation of all the segments between s and s', transforming F into other convex face (see Figure 3). Otherwise, there must exist a vertical segment s' and an oblique segment s'' in the lower part of F, because the boundary of F is a closed path. In this case, we increase the length of s, s' and s'' and we make a translation of the rest of the segments (see Figure 3).

According to the above remark, we suppose in the rest of the proof that there exists a segment parallel to s in the opposite part of F.

Suppose now that I_G has two convex faces F and F' sharing the segments $s_1, \ldots, s_n$. First of all, by the convexity of F and F', observe that if $s_1, \ldots, s_n$ are segments of P_1 (P_2), then $s_1, \ldots, s_n$ form part of P'_3 (P'_4, respectively). Thus, suppose that $s_1, \ldots, s_n$ belong to P_1 in F and to P'_3 in F'. If s is in the upper part of F, we seek another horizontal segment s' in the lower part of F and we proceed as before, increasing the length of s and s'.

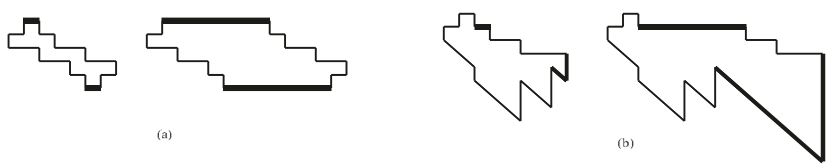


Fig. 3. How to enlarge a segment of a convex face by any amount, (a) using parallel segments or (b) using three segments.

Obviously, if s is not one of the s_i, the transformation in F does not affect the segments of F'. But, if s is one of the s_i then, on account of the first case, we consider a horizontal segment s' in the lower part of F and a horizontal segment s'' in the upper part of F'. Increasing the length of s, s' and s'', and making a translation of all the segments of F between s and s', and all the segments of F' between s and s'', we obtain a new segment representation with the same convex faces as I_G.

Suppose now that I_G has n convex faces, $F_1, \ldots, F_n$, and let s be a segment of a face $F \neq F_n$, where F_n lies in the outer face of I_G, i.e., the vertices $v_1, \ldots, v_k$ of G, corresponding to the segments $s_1, \ldots, s_k$ of F_n, lie on the outer face, and $v_2, \ldots, v_{k-1}$ are of degree two. Let us consider the graph $G_1 = G - \{v_2, \ldots, v_{k-1}\}$. Removing the segments of I_G corresponding to $v_2, \ldots, v_{k-1}$, we obtain a segment representation I_{G_1} of G_1.

Since I_{G_1} has $n-1$ faces, we can transform its segment representation into another representation which has the same convex faces, enlarging in any amount some of the segments. To obtain this new representation, we have to enlarge other segments in I_{G_1} besides s, but we have preserved the structure of the segments in the representation, so we are able to represent again the segments corresponding to $v_2, \ldots, v_{k-1}$ enlarging the length of one (or two) of them if necessary, obtaining a segment representation of G. □

3 Triangle-Free Graphs

Using the results of Barnette in [1], it can be deduced easily that every planar triangle-free graph which is a subdivision of a 3-connected graph can be reduced, by deleting edges and paths with internal vertices of degree two, to a subdivision of the complete graph K_4 in such a way that in each step we have a subdivision of a 3-connected graph. Moreover, if we fix a subgraph of G which is a subdivision of K_4, G can be reduced, by deleting edges and paths, to this subgraph. This result will allow us to build a segment representation in three directions of subdivisions of 3-connected graphs as follows. Firstly we build a segment representation of a subdivision of K_4 and then insert, in reverse order, the edges and paths that were removed to obtain from G the subdivision of K_4.

In order to do this, we need two basic operations: (1) insert a path between two segments of the same face and (2) join two segments of the same face. The second operation is, at least, as difficult as the first one, so we will concentrate only in the second operation.

We can see, using Lemma 1, that by enlarging some segments, a segment representation with convex faces can be transformed into another one preserving the topology of the embedding. But this is not sufficient to carry on the second operation, so we need another kind of transformation. Fixing an index i and changing the drawing of the segments of the path P_i of a face F, it is possible to draw a new convex face F' where the segments that were in P_i belong now to P'_{i+1} or P'_{i-1}. The rest of the paths of F' only change in the length of some of the segments (see Figure 4).

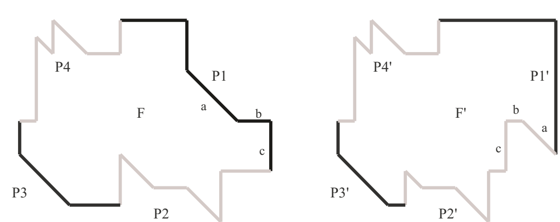


Fig. 4. How to restructure a face through the path P_1.

In the segment representation, the change of the face F to F' produces some changes in the faces adjacent to F along the path P_i, and we obtain another segment representation with convex faces as in the proof of Lemma 1 (see Figure 5). When we change the drawing in the manner described above, we say that the face F is *restructured* through the path P_i.

We need at this point some new definitions. Given a 3-colored plane graph G with colors $\{h, o, v\}$, a path $P = \{u_1, \dots, u_n\}$ of G is *rare* if it verifies one of the following conditions:

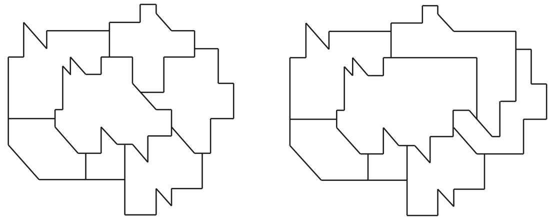


Fig. 5. How to restructure the faces adjacents to the restructured face F.

Case 1. u_1 is colored as h, u_2 as o, and u_n as v.
Case 2. u_1 is colored as v, u_{n-1} as h, and u_n as o.
Case 3. u_1 is colored as h, u_2 as o, u_{n-1} as h, and u_n as o.
Case 4. u_1 is colored as o, u_2 as v, u_{n-1} as v, and u_n as h.

A face of a graph is called *rare* if there exists a path P_i, with $i = 1, \ldots, 4$, containing a rare subpath. We say that a rare face is *repaired* if we subdivide its rare path with a new vertex colored as v between u_1 and u_2 in the first and the third case, and between u_{n-1} and u_n in the second case, or a new vertex colored as h between u_1 and u_2 in the fourth case.

On the other hand, notice that the vertical segments (except the last segment of P_1 and the last of P_3) can be extended to the interior of a convex face F, the horizontal segments too if they are in P_1 or P_3 (except the first of P_1 and the first of P_3), and the oblique segments if they are in P_2 or P_4 (except the first one of P_1 and P_3, and the last one of P_1 and P_3). Moreover, the horizontal segments in P_2 or P_4 can be extended to the interior of F if there is a vertical segment to the right or an oblique segment to the left of them; and the oblique segments in P_1 or P_3 can be extended to the interior of F if there is a horizontal segment to the right or a vertical to the left of them (see Figure 2). So, restructuring the paths and applying Lemma 1, we can transform the face F in such a way that these transformations allow us to extend two segments inside F to produce a new adjacency between them.

In the following lemma, when we say that two segments s and t can be joined by a path length k, it means a path $s, s_1, s_2, \ldots, s_k, t$.

Lemma 2. *Let I_G be a segment representation of a plane graph G and let s_1 and s_2 be two distinct segments in a convex non-rare face F. Then I_G can be transformed into another segment representation I'_G satisfying*

1. *I_G and I'_G have the same faces and face boundaries as G, and the faces of I'_G are convex;*
2. *s_1 can be joined to s_2 by a segment path of length k, for all $k > 0$, or directly if s_1 and s_2 have not the same direction inside the convex face F', corresponding to F.*

Proof. The proof falls naturally into two cases: the segments have the same direction, or they do not. In both cases, we have divided the proof in a sequence of subcases.
Case 1: s_1 and s_2 have not the same direction.
In this case, we can suppose that s_1 is vertical and s_2 is horizontal (similar arguments apply to the other configurations on the directions of the segments s_1 and s_2). The subcases are the following:

1. s_1 and s_2 are in the same path P_i.
   a) s_1 and s_2 are in P_2 (respectively in P_4).
      i. s_1 appears (clockwise) before s_2. If an oblique segment precedes s_2, by Lemma 1 the segments can be joined directly. Otherwise, the face must be restructured transforming P_2 into P_3' (respectively, P_4 into P_1').
      ii. s_2 appears (clockwise) before s_1. Since the face is not rare, s_2 precedes a vertical segment. Then, by Lemma 1, the segments can be joined directly.
   b) s_1 and s_2 are in P_1 (respectively in P_3). If s_1 appears (clockwise) before s_2, by Lemma 1 the segments can be joined directly. If s_2 appears (clockwise) before s_1, the face must be restructured transforming P_1 into P_2' (respectively, P_3 into P_4').
2. s_1 and s_2 are in different paths. If s_2 is in P_2 (respectively in P_4) and it precedes an oblique segment and s_1 is in P_3 (respectively P_1), the face must be restructured transforming P_2 into P_1' (respectively, P_4 into P_3'). Otherwise, by Lemma 1 the segments can be joined directly.

Case 2: s_1 and s_2 have the same direction.
In this case, it suffices to join the segments by a path of length 1, because this path can be substituted by any other of length greater than 1. Again there are several subcases:

1. Both are vertical segments. In this case, the segments can be joined by a path of length 1 directly using Lemma 1.
2. Both are horizontal segments. If the segments are in P_2 (respectively in P_4) and the first one precedes an oblique segment, and a vertical segment precedes the second one, then the face must be restructured transforming P_2 into P_3' (respectively, P_4 into P_1'). Otherwise, using Lemma 1 the segments can be joined directly.
3. Both are oblique segments. If the segments are in P_1 (respectively in P_3), the first one precedes a vertical segment, and a horizontal segment precedes the second one, then the face must be restructured transforming P_1 into P_2' (respectively, P_3 into P_4'). Otherwise, they can be joined directly by Lemma 1.

□

Now we prove:

Lemma 3. *Any triangle-free plane graph which is a subdivision of a 3-connected graph can be represented by segments in three directions.*

Proof. As G is a triangle-free plane graph, in virtue of Grötzsch's Theorem [9], G admits a 3-coloring with the colors h, v and o (horizontal, vertical and oblique, respectively). Since G is a subdivision of a 3-connected graph, we can find a vertex, not in the outer face of G, connected by three disjoint paths to three vertices in the outer face. These paths and the outer face determine a graph K, which is a subdivision of K_4.

Using Barnette's results in [1] it is possible to build a sequence of graphs $G_1, \ldots, G_n$ and a sequence of paths $Q_1, \ldots, Q_n$ such that $G_1 = G$, G_i is a subdivision of a 3-connected plane graph, G_i is obtained from G_{i-1} by deleting the path Q_{i-1}, and the graph obtained from G_n deleting Q_n is K.

When the path Q_i is deleted, a new face F appears in G_{i+1} as the union of two faces in G_i. The boundary of F can be divided into two paths; on the one hand the path beginning in the first vertex of Q_i and ending in the last one and, on the other hand, the rest of the boundary. If one of them is rare, we must repair F subdividing an edge with a new vertex (that we label as a *repaired* vertex), to make the face F non-rare. So, we can construct another sequence $G'_1, \ldots, G'_n$ where G'_i is G_i with a new repaired vertex, if necessary.

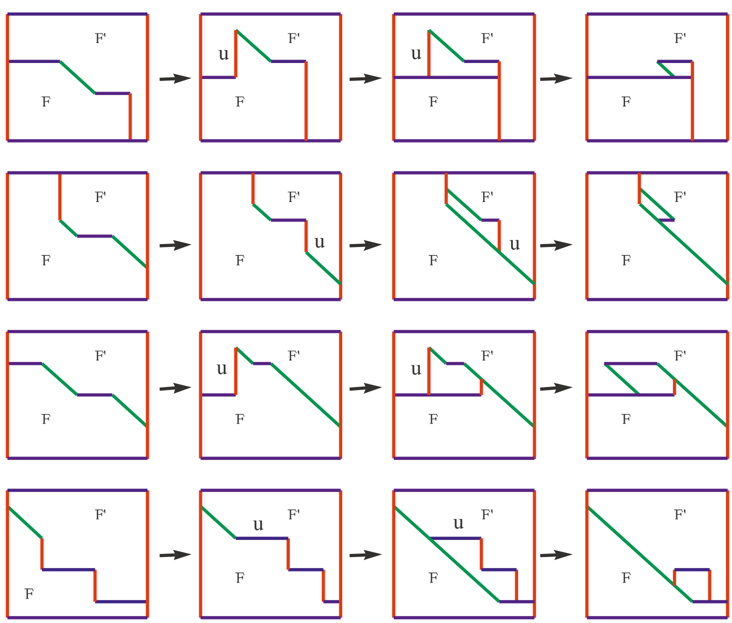


Fig. 6. How to remove the repaired vertex u in the four cases.

It is easy to give a segment representation of K' with all its faces convex. It suffices to represent convexly the outer face of K' and proceed according to the

above remark. Using Lemma 2 it is possible to add the path Q_i to the segment representation of G'_{i+1} to obtain a convex face representation of G'_i. In order to obtain a segment representation of the graph G, we must remove the repaired vertices. These vertices cannot be removed directly, so it is necessary to modify the representation. The four possible cases are illustrated in Figure 6.

Notice that we could not remove the repaired vertices if the rare path would have exactly three segments u_1, u_2 and u_3, but this case it is not possible because joining u_1 with u_3 it will be form a triangle, and G was a triangle-free graph. □

We can now formulate our main result as follows:

Theorem 1. *Every triangle-free planar graph is the intersection graph of a set of segments in three directions.*

Proof. Let G be a triangle-free plane graph. Since G has no triangles, we can obtain a 3-coloring of G using Grötszch's Theorem [9]. The colors will be labeled as h, v and o (horizontal, vertical and oblique, respectively). We can build a new triangle-free plane graph G_1, subdivision of a 3-connected graph, which contains G as a subgraph, adding new vertices and edges joining the blocks of G, possibly subdividing some edges of the blocks of G using new vertices. If this is the case, these vertices are labeled as *dummy* vertices. When an added edge produces a triangle, we subdivide it, and when a new edge joins two vertices with the same color, we subdivide it too. We call these new vertices and edges *virtual.* All the new vertices (dummy and virtual) can be colored so that the 3-coloring is preserved .

By Lemma 3, G_1 admits a segment representation. Out of this segment representation we must remove all the vertices and edges added.

A virtual edge (or a path built with virtual edges and vertices) in the segment representation of G_1 is an adjacency between two segments (or a virtual path joining two segments). It suffices to break this adjacency and to shorten these segments (note that the segments do not cross, they only contact). The dummy vertices are removed as the repaired vertices in Lemma 3. □

4 Concluding Remarks

The hypothesis that the graph has no triangles can be relaxed in some cases. No doubt there exist planar graphs with triangles that admits segment representations (see Figure 1). The problem lies in the fact that we have followed a constructive proof to obtain a segment representation using the convex faces representation of a subdivision of a 3-connected graph. This construction would not be possible in general if the graph contains triangles, because if we fix the three directions of the segments we can observe that the graph in Figure 7 cannot be represented by non-crossing segments.

In this example we see that a segment representation contains faces that do not correspond to faces of the embedding of the graph. So, the topology of the embedding of the plane graph cannot be preserved. This problem admits a

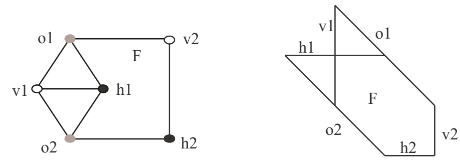


Fig. 7. Two segments must cross.

partial solution as follows. A 3-coloring of a plane graph G with colors $\{h, v, o\}$ is *good* if all the triangles of G are colored, in clockwise order, as $h - v - o$.

The proof of Theorem 1 can be adapted yielding the following result:

Theorem 2. *Let G be a 3-colored plane graph. The coloring is good if and only if exists a segment representation I_G verifying that the faces of G correspond to faces of I_G, and the boundaries of the faces are preserved.*

References

1. D.W. Barnette, "On Steinitz's theorem concerning convex 3-polytopes and on some properties of planar graphs", *The many facets of graph theory. Lectures Notes in Mathematics* Vol. 110, Springer, Berlin, pp. 27-39, 1969.
2. I. Ben-Arroyo Hartman, I. Newman and R. Ziv, "On grid intersection graphs", *Discrete Math.* Vol. 87 , pp. 41-52, 1991.
3. H. de Fraysseix, P.Osona de Mendez and J. Pach, "Representation of planar graphs by segments", *Colloquia Mathematica Societatis János Bolyai, Intuitive Geometry*, Szeged (Hungary), 1991.
4. M.C. Golumbic, "Algorithmic Graph Theory and Perfect Graphs", *Academic Press*, 1980.
5. J. Kratochvíl, "A special planar satisfiability problem and a consequence of its NP-completeness", *Discrete Applied Math.*, Vol. 52, pp. 233-252, 1994.
6. W. Naji, "Reconnaisance des graphes de cordes", *Discrete Math.* Vol. 54, pp. 329-337, 1985.
7. E.R. Scheinerman, "Intersection classes and multiple intersection parameters of graphs", *Ph D. thesis,*, Princeton University, 1984.
8. J. Spinrad, "Recognition of circle graphs", *J. of Algorithms*, Vol. 16, pp. 264-282, 1994.
9. C. Thomassen, "Grötzsch's 3-colour theorem and its counterparts for the torus and the projective plane", *J. Comb. Theory B* Vol. 62, pp. 268-279, 1994.

A Force-Directed Algorithm that Preserves Edge Crossing Properties

François Bertault

Department of Computer Science and Software Engineering
University of Newcastle
Callaghan 2308 NSW Australia
francois@cs.newcastle.edu.au

Abstract. We present an iterative drawing algorithm for undirected graphs, based on a force-directed approach, that preserves edge crossing properties. This algorithm insures that two edges cross in the final drawing if and only if these edges crossed on the initial layout. So no new edge crossings are introduced. We describe applications of this technique to improve classical algorithms for drawing planar graphs and for interactive graph drawing.

1 Introduction

Force-directed algorithms are commonly used for graph drawing because they are easy to implement and often achieve good results. Many force-directed algorithms have been proposed, which differ in the force model or the convergence method used. Usually, the force model is chosen to try to obtain uniform edge lengths, and show symmetries in the graph [2,3,4]. However, these algorithms can introduce a lot of edge crossings, which reduces the readability of the drawing. Algorithms with more complicated force models that try to reduce these edge crossings have also been proposed. These algorithms use genetic methods or simulated annealing, but are slow, difficult to parameterize, and don't insure that the resulting drawing is planar when the graph is planar.

The best classical algorithms for drawing straight-line planar graphs requires the graph to be biconnected [1,5,7]. For drawing general planar graphs, the first step is to modify the graph, by using augmentation techniques [6], that is by adding nodes and edges to the graph. Figure 1 shows an example of a planar graph, drawn using a planar graph and a basic augmentation technique. Note the lack of symmetry and variation in edge length.

The basis of the approach presented in this paper is to combine the useful characteristics of both classical planar graphs and force-directed algorithms. The approach proceeds by first finding a planar drawing using classical algorithms and then iteratively applying a new force directed algorithm, called `PrEd`. The final drawing improves symmetry and uniformity of edge lengths, while preserving initial edge-crossing properties of the graph. This insures that no new edge crossings are introduced. Figure 1 shows an example of planar graph drawn with

J. Kratochvíl (Ed.): GD'99, LNCS 1731, pp. 351–358, 1999.

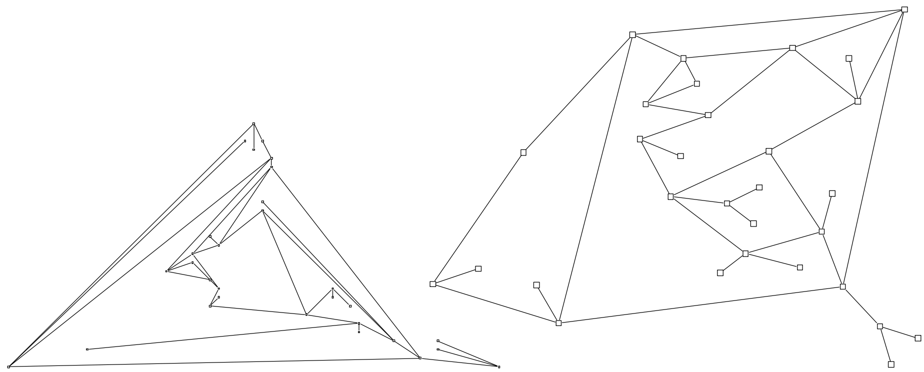


Fig. 1. Left: planar graph obtained using the LEDA planar graphs drawing algorithm. Right: planar graph obtained by applying the `PrEd` algorithm to the initial layout on the left. Note that the planar embedding remains the same.

this method. The initial planar layout is shown in Fig. 1. Note that the planar embedding remains the same.

The `PrEd` algorithm is not limited to planar graphs. The intrinsic property of the algorithm is that two edges cross on the final drawing if and only if these edges crossed on the initial layout. This property is well suited for interactive graph drawing. A user can easily indicate the desired relative position between nodes and edges by simply moving nodes interactively. The `PrEd` algorithm will then improve the graph symmetry and produce more uniform edge lengths.

2 `PrEd` algorithm

We consider a graph $G = (V, E)$, where V is a set of nodes, and E a set of undirected edges $(a, b) = (b, a) \in V \times V$. The number of nodes is denoted $|V|$ and the number of edges $|E|$. The principle of the `PrEd` algorithm is the following. At each iteration, for each node v of the graph, a force $F(v)$ is computed, that depends on the positions of the nodes and edges of the graph. Each node is then moved in the direction of $F(v)$. The main difference with classical force-directed algorithms is that we restrict, for each node, the maximum amplitude of the move such that the edge-crossing properties are preserved. This restriction depends on the direction of the force.

2.1 Computation of forces

The force model is very similar to other force-directed algorithms. Three kinds of forces between nodes and edges are considered: the attraction forces between nodes linked by an edge; the repulsion forces between each pair of nodes, and the repulsion forces between nodes and edges.

The position of a node v is denoted $(x(v), y(v))$. The Euclidean distance between two nodes a and b is denoted $d(a,b)$ and δ is the length of the edges that we would like to obtain. The force applied to a node v is denoted $F(v) = (F_x(v), F_y(v))$. The attraction force $F^a(u,v)$ and repulsion force $F^r(u,v)$ between two nodes u and v of the graph, are defined by:

$$F_x^a(u,v) = \frac{d(u,v)}{\delta}(x(v) - x(u)) \qquad F_x^r(u,v) = \frac{-\delta^2}{d(u,v)^2}(x(v) - x(u))$$

This has been shown to be effective [3].

For the computation of the repulsing force $F^e(v,(a,b))$ between a node v and an edge (a,b), we consider a virtual node i_v, defined by the projection of the node v on the vector formed by the vertices of the edge. A force is applied to nodes v, a and b only if the virtual node i_v is located on the edge, and if the distance between v and i is smaller than a parameter γ. We ignore forces if $v = a$ or $v = b$.

$$F_x^e(v,(a,b)) = \begin{cases} -\dfrac{(\gamma - d(v,i_v))^2}{d(v,i_v)}(x(i_v) - x(v)) & \text{if } i_v \in (a,b), d(v,i_v) < \gamma \\ & \quad v \neq a, v \neq b \\ 0 & \text{otherwise} \end{cases}$$

The overall force applied to a node v is obtained by summing the attraction and repulsion forces.

$$F_x(v) = \sum_{(u,v)\in E} F_x^a(u,v) + \sum_{u\in V} F_x^r(u,v) + \sum_{(a,b)\in E} F_x^e(v,(a,b)) - \sum_{\substack{u\in V, w\in V \\ (v,w)\in E}} F_x^e(u,(v,w))$$

We have a similar formula for the $F_y(v)$ component.

2.2 Computation of the amplitude of moves

The node-edge repulsion force has been used in other algorithms in order to avoid overlapping between nodes and edges. However, since we consider discrete moves of the nodes during the steps of the algorithm, this does not guarantee that a node will not cross an edge and create a new crossing. For preserving the crossing properties between edges, a *zone* $Z(v)$ is associated to each node v. $Z(v)$ indicates where the node v is allowed to move. A zone is defined by eight arcs $Z_1(v), \ldots, Z_8(v)$. The zone of a node v can be represented with only eight values $R_1(v), \ldots, R_8(v)$, corresponding to the radius of the arcs. The zone of a node is not the maximal area in which a node is allowed to moved without changing the crossing properties. The choice of representing zones by eight arcs is a good compromise between efficiency and liberty of move of the nodes.

The amplitude of move of a node v, with a force $F(v)$ applied to it, is bounded by the value of the arc that contains $F(v)$. For example, in Fig. 2, the node v moves in direction of the force $F(v)$, with an amplitude bounded by $R_8(v)$. An arc with an infinite radius is represented in white on the figure.

The computation of the zones is made such that we avoid the creation of new crossings, while considering only one node and one edge at a time. For each pair of node v and edge (a, b), the idea is to consider two cases, depending on the position of the virtual node i_v that we defined for the computation of repulsive forces between nodes and edges. In both cases, we restrict the move of the nodes v, a and b for avoiding the creation of crossing. If the virtual node i_v is located on the edge (a, b), we also allow the node v to "escape" by increasing its distance to the edge (a, b). If the virtual node i_v is not located on the edge (a, b), we allow the node v to "turn around" the edge (a, b).

To compute the zones of each node of the graph, the eight values of each zone are first initialized to infinity. Then, for each pair of node v and edge (a, b), we consider the position of the virtual node i_v:

Case 1 If i_v is on the edge (a, b) (Fig. 3). We consider the segment $[v, i_v]$, and we search which arc s of $Z(v)$ intersects the segment. We update the values of the zones as follow:

$$\begin{aligned}
R_j(v) &= \min(R_j(v), d(v, i_v)/3), &\quad j &= r(s-2), \ldots, r(s+2)\\
R_j(a) &= \min(R_j(a), d(v, i_v)/3), &\quad j &= r(s+2), \ldots, r(s+6)\\
R_j(b) &= \min(R_j(b), d(v, i_v)/3), &\quad j &= r(s+2), \ldots, r(s+6)
\end{aligned}$$
$$\text{where } r(j) = 1 + (j \mod 8)$$

Case 2 If i_v is not on the edge (a, b) (Fig. 4). We update the values of the zones as follow:

$$\begin{aligned}
R_j(v) &= \min(R_j(v), \min(d(a, v), d(b, v))/3), &\quad j &= 1, \ldots, 8\\
R_j(a) &= \min(R_j(a), d(a, v)/3), &\quad j &= 1, \ldots, 8\\
R_j(b) &= \min(R_j(b), d(b, v)/3), &\quad j &= 1, \ldots, 8
\end{aligned}$$

In summary, the steps of the `PrEd` algorithm are, for one iteration:

Step 1 Computation of the forces applied to each node: $O(|V|^2 + |V||E|)$.
Step 2 Computation the values of the zone of each node: $O(|V||E|)$.
Step 3 Move of each node, with the amplitude of the move bounded by its zone: $O(|V|)$.

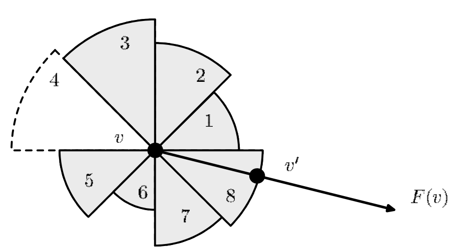


Fig. 2. Move of a node v according to its zone $Z(v)$.

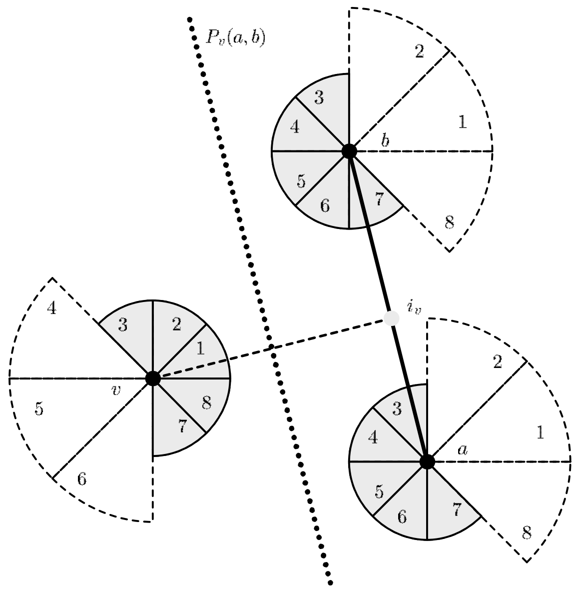


Fig. 3. Computation of the zones for a node v and an edge (a, b) (case 1).

The overall complexity of one step is $O(|V|^2 + |V||E|)$. In practice, the algorithm works well for small or sparse graphs. For example, the figure 1 was obtained in 100 iterations and 3 seconds. In some cases, the distance between nodes can be small.

2.3 Correctness

Theorem 1. *The* **PrEd** *algorithm preserves edge-crossing properties of a graph.*

Proof. We first show that one step of the algorithm preserves non crossing edges. For this, we consider two non crossing edges (u, v) and (a, b). The idea of the proof is to show that we can construct two polytopes (i.e. an intersection of half planes), $P(u, v)$ and $P(a, v)$, such that we have:

$$P(u,v) \cap P(a,b) = \emptyset,\ Z(u) \cup Z(v) \subset P(u,v) \text{ and } Z(a) \cup Z(b) \subset P(a,b) \quad (2.1)$$

Since a polytope is convex, and the ends of edges can only move inside their zone, the edges remains in their polytope after the moves of the ends.

The key point of the proof is to show, for a node v and an edge (a, b), that we can define a polytope $P_v(a, b)$, containing $Z(a)$ and $Z(b)$ and not $Z(v)$. We consider the two following cases, according to the position of the virtual node i_v. The other cases are deduced by renaming:

Case 1 If $i_v \in (a, b)$. We define $P_v(a, b)$ by the half plane with a boundary orthogonal to the segment $[v, i_v]$, that passes throw the middle of this segment as shown in Fig. 3.

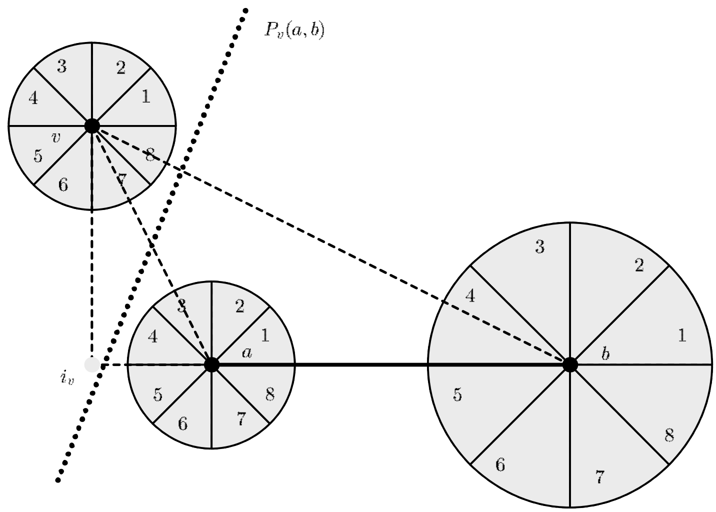


Fig. 4. Computation of the zones for a node v and an edge (a, b) (case 2).

Case 2 If $i_v \notin (a, b)$, with $d(i_v, a) < d(i_v, b)$. We define $P_v(a, b)$ by the half plane with a boundary forming an angle of $\pi/4$ with the segment $[a, v]$, as shown in Fig. 4. This plane contains $Z(a)$ if the radius of $Z(a)$ is smaller than $d(a, v)/(2\sqrt{2})$, and we chose to define the radius of the zones by $d(a, v)/3$. We also verify easily that this half plane contains also $Z(b)$, since the radius of $Z(b)$ is given by $d(b, v)/3$.

The final step of the proof is to verify, according to the positions of the virtual nodes, that we can define the polytopes $P(u, v)$ and $P(a, b)$ that verify property (2.1).

Case 1 If $i_u \in (a, b)$ and $i_v \in (a, b)$: we define $P(a, b) = P_u(a, b) \cap P_v(a, b)$, and $P(u, v) = \mathbb{R}^2 \setminus P(a, b)$.

Case 2 If $i_u \in (a, b)$, $i_v \notin (a, b)$:
- (a) If $i_a \in (u, v)$, $i_b \notin (a, b)$, and $d(u, i_u) < d(a, i_a)$: we define $P(a, b) = P_u(a, b)$, and $P(u, v) = \mathbb{R}^2 \setminus P(a, b)$.
- (b) If $i_a \notin (u, v)$ and $i_b \notin (a, b)$: we define $P(a, b) = P_u(a, b)$, and $P(u, v) = \mathbb{R}^2 \setminus P(a, b)$.

Case 3 If $i_u \notin (a, b)$, $i_v \notin (a, b)$, $i_a \notin (u, v)$ and $i_b \notin (a, b)$, with $d(a, u) \leq \min(d(a, v), d(b, u), d(b, v))$: we define $P(a, b) = P_u(a, b)$, and $P(u, v) = \mathbb{R}^2 \setminus P(a, b)$.

For verifying that one step of the algorithm preserves crossing edges, we use similar techniques. If two edges (u, v) and (a, b) cross, we remark that we can remove the crossing by moving the nodes only if we can obtain a crossing between the virtual edges (u, a) and (v, b), or between (u, b) and (v, a).

3 Further work

The efficiency of the algorithm could be greatly improved. We could, instead of considering all edges of the graph for defining the zone of a node, consider only a restricted set of edges that “surround” that node. The set of edges that we need to consider could be determine in a preprocessing step of the algorithm.

The PrEd algorithm can also be combined with a simple “jumping” heuristic, for reducing edge-crossings in general graph drawing. The initial positions of nodes are obtained using a classical force-directed algorithm. Then, for each pair of edge crossings, we test, during the iterations of the PrEd algorithm, if the fact to remove this crossing, by placing one end near the intersection, reduces the total number of crossing in the graph. This naive approach can give good results on simple graphs, but is very expensive. Figure 6 demonstrates this algorithm to remove edge crossings. Further work could be to look for better heuristics based on this idea.

4 Conclusion

The PrEd algorithm takes $O(|V|^2 + |V||E|)$ time to complete one iteration. It preserves crossing and non-crossing edges during the iteration steps. This property is well suited for improving the drawing of straight-line planar graphs, where the initial position of nodes is obtained from a classical algorithm.

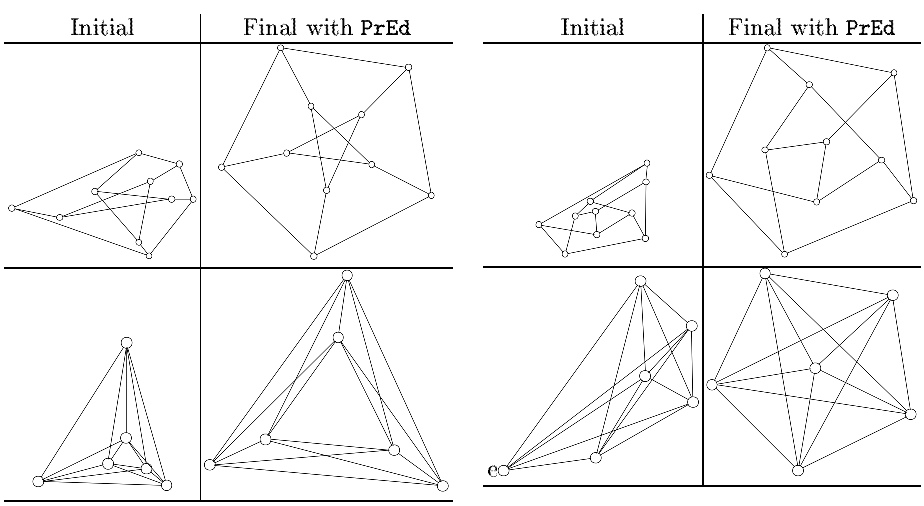


Fig. 5. Examples of interactive drawing with the PrEd algorithm. The initial position of nodes is specified by the user, and the PrEd algorithm improves symmetry and edge length criteria without changing the relative position of nodes.

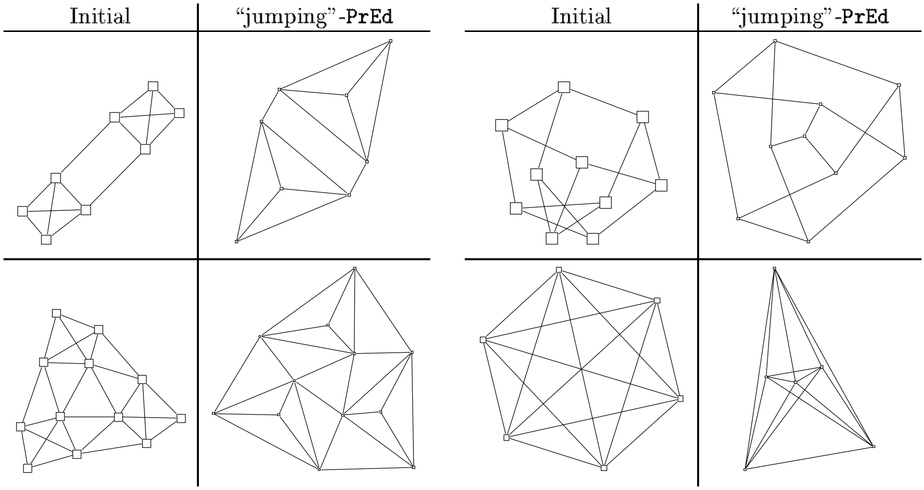


Fig. 6. Combination of a the `PrEd` algorithm with a "jumping" operation for reducing edge-crossings.

The `PrEd` algorithm can also be applied for interactive drawing of general graphs. The user can modify interactively the position of the nodes, and the algorithm improves the drawing criteria such as symmetry and uniform edges lengths. Relative positions of nodes are preserved by the algorithm. Small modifications of the graph induces small modifications on the new representation of the graph. Examples of drawings obtained this way are shown in Fig. 5.

References

1. H. de Frayssex, J. Pach, and R. Pollack. How to draw a planar graph on a grid. *Combinatorica*, 10:41–51, 1990.
2. P. D. Eades. A heuristic for graph drawing. *Congressus Numerantium*, 42:149–160, 1984.
3. T. Fruchterman and E. Reingold. Graph drawing by force-directed placement. *Software-Practice and Experience*, 21(11):1129–1164, 1991.
4. T. Kamada and S. Kawai. An algorithm for drawing general undirected graphs. *Information Processing Letters*, 31:7–15, 1989.
5. G. Kant. Drawing planar graphs using the *lmc*-ordering. *Proc. IEEE Symp. on Foundation of Computer Science*, pages 101–110, 1992.
6. Goos Kant and Hans L. Bodlaender. Triangulating planar graphs while minimizing the maximum degree. *Information and Computation*, 135(1):1–14, 1997.
7. P. Mutzel. A fast linear time embedding algorithm based on the Hopcropt-Tarjan planarity test. Technical report, Institut für Informatik, universität zu Köln, 1992.

Rectangle of Influence Drawings of Graphs without Filled 3-Cycles*

Therese Biedl[1]**, Anna Bretscher[2], and Henk Meijer[2]

[1] Department of Computer Science University of Waterloo
Waterloo, Ontario N2L 3G1, Canada
biedl@uwaterloo.ca

[2] Computing and Information Science Department Queen's University
Kingston, Ontario K7L 3N6, Canada
{bretscha,henk}@qucis.queensu.ca

Abstract. In this paper, we study rectangle of influence drawings, i.e., drawings of graphs such that for any edge the axis-parallel rectangle defined by the two endpoints of the edge is empty. Specifically, we show that if G is a planar graph without filled 3-cycles, i.e., a planar graph that can be drawn such that the interior of every 3-cycle is empty, then G has a rectangle of influence drawing.

1 Introduction

Let $G = (V, E)$ be a planar graph. A *planar straight-line drawing* of G is a mapping of the vertices to points in the two-dimensional grid such that no two edges intersect (except at common endpoints) and no edge intersects a vertex other than its endpoints. It has been known for a long time that every planar graph has a planar straight-line drawing; see e.g. [6]. In this paper, we are interested in straight-line drawings that are also *proximity drawings*: an edge in the drawing implies that the two endpoints of the edge are "close" by some definition. In this paper, we study a type of proximity drawings known as *rectangle of influence drawings*, where an edge implies that the axis-parallel rectangle defined by its endpoints contains no other points of the drawing.

Existing results A recent survey of proximity drawings can be found in [5]. The notion of rectangular visibility where two points "see" each other if and only if their rectangle of influence is empty, was first defined in [7]. Alon et al. call two points box separated if and only if their closed rectangle of influence is empty. They constructed an upper bound on the graph of all box separated points, i.e. on the closed rectangle of influence graph [1]. Rectangle of influence graphs were then investigated in [9] with respect to the data analysis problem and later extended by [4] such that two sets of points are given, P and O, in

* The results are part of the second author's Master's thesis at Queen's University.
** Research partially supported by NSERC. Part of this research was done while the author was at McGill University.

J. Kratochvíl (Ed.): GD'99, LNCS 1731, pp. 359–368, 1999.

which $p_1, p_2 \in P$ see each other if and only if their rectangle of influence contains no points from O.

In [8], Liotta et al. examined rectangle of influence drawings with respect to graph characterization or the proximity drawing problem. In particular, they investigate whether several subclasses of planar graphs as well as complete graphs admit rectangle of influence drawings. They study both the model where the rectangle of influence is closed (which is what we mainly consider in this paper), and also the model where the rectangle of influence is open.

In all existing works, only *strong* rectangle of influence drawings have been studied; i.e., there is an edge (v, w) in the drawing *if and only if* the axis-parallel rectangle defined by the points assigned to v and w is empty except for these two points. As opposed to this, we will allow *weak* rectangle of influence drawings, where edges are optional, i.e., need not be added if the rectangle of influence is empty.

Our results We study weak planar rectangle of influence drawings and show that a large class of planar graphs has such a drawing. More precisely, if G is a planar graph that has a planar drawing such that the interior of any 3-cycle is empty (we call this a *drawing of G without filled 3-cycle*), then G has a weak planar rectangle of influence drawing. Since in any planar closed rectangle of influence drawing, any 3-cycle must be empty, the converse also holds: if G is a planar graph, then it has a weak planar closed rectangle of influence drawing if and only if it can be drawn without filled 3-cycle.

Note that a drawing without filled 3-cycle is the same as a drawing where the outer-face is not a 3-cycle and that has no *separating 3-cycle* (that is, a 3-cycle such that both inside and outside the cycle are other vertices). Testing whether a graph can be embedded without a separating 3-cycle can be done in $O(n)$ time [2], and a slight modification of this algorithm yields an $O(n)$ time test whether a planar graph has a drawing without filled 3-cycles.

2 Definitions

We assume familiarity with basic graph theoretic and geometric terminology (see also [3] and [10]).

A graph is called *planar* if it can be drawn in the plane without an edge crossing. A specific planar drawing splits the plane into components called *faces*; the unbounded face is called the *outer-face*. A planar graph is called *triangulated* if all faces are 3-cycles, and *internally triangulated* if all faces except the outer-face are 3-cycles.

If a planar drawing is fixed, then we call a 3-cycle *filled* if there is a vertex inside the drawing of the 3-cycle. If G is a planar graph that has a planar drawing such that there are no filled 3-cycles, then we call G an *NF3-graph*.

A *straight-line embedding* of a graph $G = (V, E)$ is an injective mapping $p : V \to R^2$ such that for any edge (v, w) and any vertex x, the open line segment $(p(v), p(w))$ does not intersect $p(x)$. It is called a *planar straight-line embedding* if additionally for any pair of edges $(v, w), (v', w')$ the open line segments $(p(v), p(w))$ and $(p(v'), p(w'))$ do not intersect. For ease of notation, we

will drop the use of the function p, and use graph-theoretic terms such as vertex to denote both the graph-theoretic object and its representation in a drawing.

Given two distinct points p, p' in the plane, $R[p, p']$ denotes the minimal axis-aligned rectangle containing both p and p'. We call $R[p, p']$ the *closed rectangle of influence of* p, p'. If $R[p, p']$ is a non-degenerate rectangle, then we denote by $R(p, p')$ the interior of $R[p, p']$. If $R[p, p']$ is a line segment, then we denote by $R(p, p')$ the open line segment between p and p'. We call $R(p, p')$ the *open rectangle of influence of* p, p'. Let P be a set of points. If we take P, together with all line segments $[p, p']$ (where $p, p' \in P$) for which the open/closed rectangle of influence contains no points of P other than p and p', then we obtain a straight-line drawing of some graph, called the *strong open/closed rectangle of influence graph of* P. We say that a graph G has a *weak open/closed rectangle of influence drawing* if there exists a one-to-one mapping $V \to P \subset R^2$ such that G is a subgraph of the strong open/closed rectangle of influence graph of P. As we will not study strong rectangle of influence drawings further, for easier notation we will drop "weak" for the remainder of this paper. We also use the term *oRI-drawing* for an open rectangle of influence drawing, and *cRI-drawing* for a closed rectangle of influence drawing.

3 Rectangle of Influence Drawings for NF3-Graphs

Our study of NF3-graphs was motivated by the following observation:

Lemma 1. *If a graph admits a planar closed rectangle of influence drawing, then it is an NF3-graph.*

Proof. Assume that a graph G has a planar cRI-drawing, and let $\{a, b, c\}$ be the points of a 3-cycle of G. The minimum enclosing rectangle of points $\{a, b, c\}$ must be empty (except for a, b, c), because it is the union of the rectangles of influence $R[a, b], R[b, c], R[c, a]$, which must be empty because these are edges. Since the minimum enclosing rectangle also contains the straight-line drawing of the 3-cycle $\{a, b, c\}$, and since the drawing is planar, that means that no vertex can be inside the 3-cycle in G. So every 3-cycle in G is empty, which means that G is an NF3-graph.

In this section, we show the reverse of this lemma, i.e., we show that every NF3-graph admits a planar closed rectangle of influence drawing. We prove this first for a special class of NF3-graphs; namely, those that are biconnected and have an embedding that is internally triangulated and has no *chord*, i.e., no edge between two non-consecutive vertices on the outer-face. Then we show that any NF3-graph can be converted into such an NF3-graph by adding edges; this shows that any NF3-graph has a weak planar rectangle of influence drawing.

3.1 Biconnected, Chordless, Internally Triangulated NF3-graphs

Assume for the remainder of this section that G is a biconnected NF3-graph and that an embedding of G exists that has no filled 3-cycles, no chords, and is

internally triangulated. This embedding will be used as the input for a drawing algorithm presented in this section and whenever we talk about the graph G, we in fact refer to this embedding. Throughout this section, we assume that G has at least 4 vertices; any graph with less than 4 vertices has a planar cRI-drawing. This implies that the outer-face has at least 4 vertices, because if it had 3 vertices, it would be a filled 3-cycle since $n \geq 4$.

Our algorithm to create a planar cRI-drawing of G is similar in spirit to previous algorithms for planar straight-line drawings (e.g. [6]): add vertices to the drawing one-by-one or in small groups, and maintain an invariant that allows one to add these vertices easily. However, both the manner of selecting the next group of vertices and the invariant is different in our algorithm.

The Invariant Our invariant is, loosely speaking, that the relevant part of the outer-face of the current subgraph is drawn on a diagonal. To make this more precise, we need more definitions. To be able to state these definitions precisely, we modify the input graph.

Definition 1. *Let G_s denote a connected induced subgraph of $G = (V, E)$ and let V_s be the set of vertices of G_s. Let $v_0, v_1, v_2, \ldots, v_{m-1}$ be the list of vertices on the outer-face of G_s in order. G_s may not be biconnected, so some vertices may appear more than once in this list. We require that any vertex in $V - V_s$ lies in the outer-face of G_s. An edge (v, w) on the outer-face of G_s is called an* active edge *if one of its incident faces is not the outer-face of G and contains a vertex from $V - V_s$. A vertex v_i on the outer-face of G_s is called an* internal vertex *if both (v_{i-1}, v_i) and (v_i, v_{i+1}) are active edges and there is path from v_{i-1} to v_{i+1} that uses only vertices in $V - V_s$ (where the arithmetic in the indices is done modulo m). A* ridge *$R = \{r_1, r_2, \ldots, r_l\}$ of G_s is a set of consecutive vertices on the outer-face of G_s such that R contains all active edges and internal vertices of G_s.*

Definition 2. *A set of* monotonically decreasing points *is a set of points $P = \{p_1, p_2, \ldots, p_l\}$ such that for $1 \leq i < l$ we have $x(p_i) < x(p_{i+1})$ and $y(p_i) > y(p_{i+1})$.*

Definition 3. *Given a subgraph $G_s \subset G$, a planar cRI-drawing Γ of G_s is said to* satisfy the invariant *if there exists a ridge $R = \{r_1, \ldots, r_l\}$ such that*

- *R is a simple path,*
- *R is drawn on a set of monotonically decreasing points, and*
- *no point in Γ is* above the ridge, *i.e., if we draw the polygonal chain through points in R, then no point in Γ is strictly above or to the right of a point of this polygonal chain.*

For an illustration of the above definitions, see Figure 1.

Remark 1. If Γ satisfies the invariant, then we may assume without loss of generality that the ridge is drawn on a diagonal line.

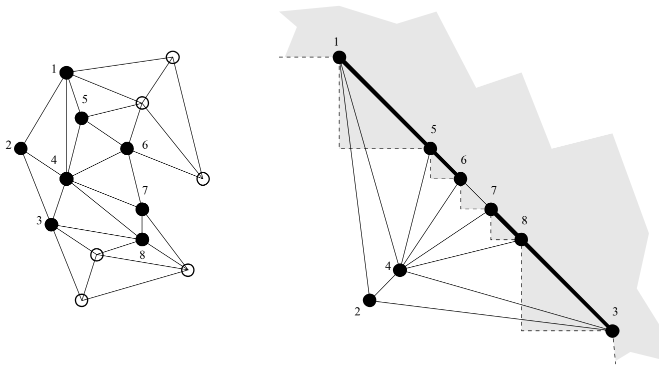


Fig. 1. Illustration of the invariant. On the left is a graph G with vertices in V_s in black; on the right is a drawing of G_s that satisfies the invariant; the ridge R consists of vertices 1, 5, 6, 7, 8 and 3; the region that is known not to contain any other point of the drawing of G_s is shaded; active edges are shown thicker and vertices 5 and 8 are internal.

Proof. Because the ridge is on a set of monotonically decreasing points, we have $x(r_1) < x(r_2) < \cdots < x(r_l)$ and $y(r_1) > y(r_2) > \cdots > y(r_l)$. It is known [8] that if we apply an *order-preserving* transformation to a cRI-drawing, i.e., a transformation so that the orders along the x-axis and along the y-axis are preserved, then the resulting drawing is again a cRI-drawing. One can also verify that if the initial drawing was planar, then so is the resulting drawing. Thus, we can define a transformation of Γ that maps r_i to $(i, l+1-i)$ and that is order-preserving. The resulting drawing then also satisfies the invariant and the ridge is drawn on a diagonal line.

Adding Vertices In this section we show how to build a drawing while maintaining the invariant. We start by initialising V_s as a set of consecutive vertices on the outer-face of G. If the outer-face of G has size t, we require that $3 \leq |V_s| < t$. G_s is the subgraph induced by V_s and the ridge R contains all vertices in V_s. Since G is biconnected, R is a simple path. We place $r_1, \ldots, r_l$ on the diagonal $y = -x$. Since l is less than the cardinality of the outer-face and G has no chords, there are no edges between these vertices other than between consecutive ones, so we obtain an cRI-drawing of G_s. Notice that all edges in G_s are active and that all vertices except r_1 and r_l are internal. It is easy to verify that the invariant holds.

So assume now that we have a drawing Γ of a subgraph G_s that satisfies the invariant, and $R = \{r_1, \ldots, r_l\}$ is the ridge of this drawing. Now we show how to add more vertices to this drawing. Specifically, we will give two methods of adding vertices that clearly maintain the invariant. In the next section we then show that one of these possibilities can always be applied.

We first develop a generic placement of a point such that the point *sees* (that is, has an empty rectangle of influence to) many points on the ridge. For a vertex

v, denote by $x(v)$ and $y(v)$ the x- and y-coordinate of the point that corresponds to v in Γ. We define

$$\begin{aligned} row_{r_i} &= \{(x,y) \in R^2 | x > x(r_{i+1}), y(r_i) < y < y(r_{i+1})\}, \text{ and} \\ col_{r_j} &= \{(x,y) \in R^2 | y > y(r_{j-1}), x(r_{j-1}) < x < x(r_j)\}. \end{aligned}$$

See also the left picture of Figure 2.

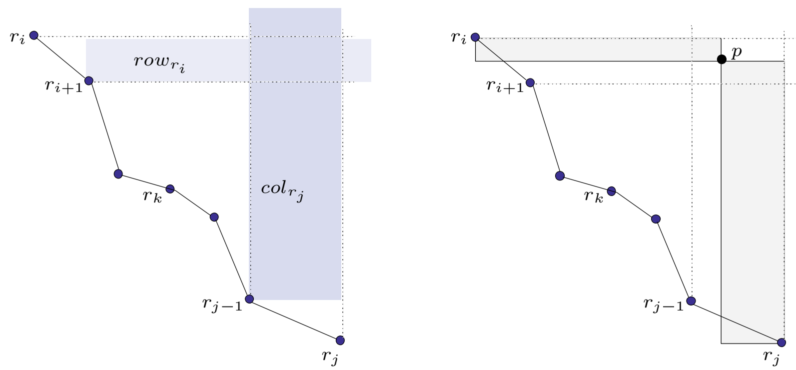


Fig. 2. Definition of row_{r_i} and col_{r_j}. Placing a point p in their intersection means that p sees all vertices $r_i, r_{i+1}, \ldots, r_j$.

Lemma 2. *If p is a point in $row_{r_i} \cap col_{r_j}$ for $i < j-1$, then $R[p, r_k]$ is empty for $i \leq k \leq j$.*

Proof. For $i < k < j$, all points in $R[p, r_k]$ are above the ridge, therefore $R[p, r_k]$ is empty by the invariant. For $k = i$, note that $R[r_i, r_{i+1}]$ is empty because (r_i, r_{i+1}) is an edge. Since all points in $R[p, r_i]$ are either above the ridge or in $R[r_i, r_{i+1}]$, it follows that $R[p, r_i]$ is empty. Similarly one shows that $R[p, r_j]$ is empty.

To explain the methods of adding vertices, we distinguish types of neighbours of the ridge-vertices.

Definition 4. *Let the* ridge degree*, or rdeg(v), of a vertex $v \in V - V_s$ be the number of ridge vertices adjacent to v.*

Definition 5. *For $1 \leq i \leq l$, let B_i be the neighbours of r_i in $V - V_s$ that have ridge degree ≥ 2 and are adjacent to a consecutive set of ridge-vertices starting at r_i. More formally, $B_i = \{v \in V - V_s :$ there exists a $j > i$ such that the neighbours of v in R are $r_i, r_{i+1}, \ldots, r_{j-1}, r_j\}$. Set $B = \bigcup_{i=1}^{l} B_i$. We call a vertex v a* vertex of type B *if $v \in B$.*

Lemma 3. *For $1 \leq i \leq l$, we have $|B_i| \leq 1$.*

Proof. Assume that we had two vertices b_1, b_2 which both belong to B_i. By the definition of G_s, this means that they are both in the outer-face of G_s. And by Definition 5 both are adjacent to r_i and r_{i+1}. One of these two 3-cycles must be filled, contradicting the assumption on the graph.

In the remainder of this section, we will denote by b_i the (unique) vertex in B_i, if it exists.

Definition 6. *For $1 \leq i \leq l$, let C_i be the vertices in $V - V_s$ that are adjacent to r_i and have ridge degree 1. Set $C = \bigcup_{i=1}^{l} C_i$. We call a vertex v a* vertex of type C *if $v \in C$.*

Definition 7. *For $1 \leq i \leq l$, let D_i be all those vertices in $V - V_s$ that are adjacent to r_i, but neither of type B nor of type C. Set $D = \bigcup_{i=1}^{l} D_i$. We call a vertex v a* vertex of type D *if $v \in D$.*

A vertex of type D must have ridge degree ≥ 2 (otherwise it would be of type C), and its neighbours on the ridge are not an interval on the ridge (otherwise it would be type B). In particular therefore, if $v \in D$, then there exists i, j such that $j > i + 1$ and v is adjacent to r_i and r_j, but not to r_{i+1}.

Placement I Assume there exists a vertex $v \in B$ such that $rdeg(v) \geq 3$.

Say, $v = b_i$, then by definition b_i is adjacent to $r_i, r_{i+1}, \ldots, r_j$ for some $j > i + 1$. Place b_i in the intersection of row_{r_i} and col_{r_j}. The new ridge is $R = \{r_1, \ldots, r_i, b_i, r_j, \ldots, r_l\}$. By Lemma 2, b_i can see all of $r_i, \ldots, r_j$, so the new drawing is a planar cRI-drawing. One easily verifies the other conditions of the invariant.

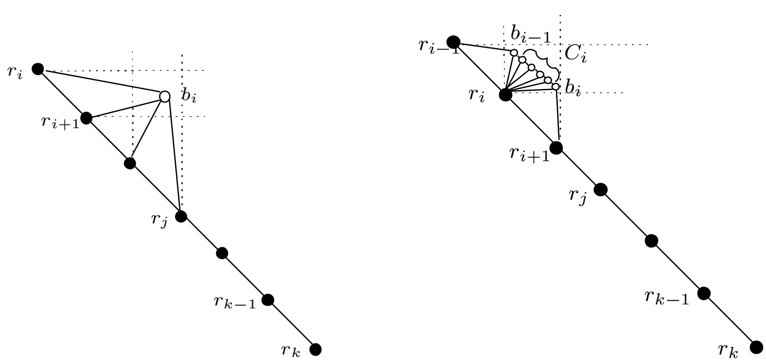


Fig. 3. Illustration of Placement I and Placement II.

Placement II Assume there exists an $1 \leq i \leq l$ such that r_i is internal and $D_i = \emptyset$.

Since r_i is internal, the two edges $e_0 = (r_{i-1}, r_i)$ and $e_1 = (r_i, r_{i+1})$ on the outer-face of G_s are active and hence on the ridge, in particular therefore $i \geq 2$. Since e_0 and e_1 are active and G is internally triangulated, there are vertices x_{i-1} and x_i in $V - V_s$ such that x_{i-1} is adjacent to r_{i-1} and r_i and x_i is adjacent to r_i and r_{i+1}.

By definition, $rdeg(x_{i-1}) \geq 2$ and $rdeg(x_i) \geq 2$. Assume first $rdeg(x_i) \geq 3$. Because $D_i = \emptyset$, x_i must be of type B or C, but by $rdeg(x_i) \geq 3$ it must be of type B. So we have found a vertex of type B with $rdeg(x_i) \geq 3$, which means that we can apply Placement I. Similarly, we apply Placement I if $rdeg(x_{i-1}) \geq 3$.

So we can place a vertex unless $rdeg(x_{i-1}) = rdeg(x_i) = 2$. This implies that x_{i-1} is incident only to r_{i-1} and r_i on the ridge, and x_i is incident only to r_i and r_{i+1} on the ridge. Therefore $x_{i-1} = b_{i-1}$ and $x_i = b_i$, and in particular $x_{i-1} \neq x_i$.

Write $C_i = \{c_1, c_2, \ldots, c_j\}$ such that $\{b_{i-1}, c_1, \ldots, c_j, b_i\}$, are the neighbours of r_i in $G - G_s$ in clockwise order; this is possible because $D_i = \emptyset$. Note that C_i may be empty, in this case we have $j = 0$. Let $b_{i-1} = c_0$ and $b_i = c_{j+1}$. Because G is internally triangulated and r_i is internal, the incident faces of r_i are 3-cycles, and therefore the edges (c_k, c_{k+1}), $0 \leq k \leq j$ must exist. Place $\{b_{i-1}, c_1, \ldots, c_j, b_i\}$ on a diagonal within the intersection of $row_{r_{i-1}}$ and $col_{r_{i+1}}$ such that they are monotonically decreasing; see the right picture of Figure 3.

Since $row_{r_{i-1}} \cap col_{r_{i+1}}$ was empty in Γ and we placed the new vertices on a monotonically decreasing diagonal, c_k can see c_{k+1} for $0 \leq k \leq j$. Also, any c_k, $0 \leq k \leq j$ can see r_i; this holds because the configuration is simply Lemma 2 rotated by π. Notice that $\{b_{i-1}, c_1, \ldots, c_j, b_i\}$ has no chord, because any chord would induce a filled 3-cycle between the two endpoints and r_i. Therefore, the new drawing is a planar cRI-drawing of the subgraph induced by the vertices. The new ridge is $R = \{r_1, \ldots, r_{i-1}, b_{i-1}, c_1, \ldots, c_j, b_i, r_{i+1}, \ldots, r_l\}$. One can verify the conditions of the invariant.

Proof of Correctness We now show that Placement I or Placement II is always possible. In fact, we show that Placement II (which possibly causes a Placement I) is always possible.

Lemma 4. *Let G be a biconnected, chordless, internally triangulated NF3-graph. Let G_s be a strict subgraph of G, and let Γ be a planar cRI-drawing of G_s that satisfies the invariant with ridge $R = \{r_1, \ldots, r_l\}$. Then there exists a vertex r_i that is internal and $D_i = \emptyset$.*

Proof. We have two cases.

Case 1: $D = \emptyset$. Since G_s is a strict subgraph of G, there exists a vertex v on the outer-face of G_s that has neighbours in $G - G_s$. At least one incident edge of v on the outer-face of G_s must be active, for if they were both inactive, then v would be a cut-vertex of G, but G is biconnected by assumption.

So assume $e = (v, w)$ is active. If neither v nor w are internal, then e would be a chord of G, which is impossible because G is chordless. So assume that one vertex, say w, is internal and by the invariant it belongs to the ridge, say $w = r_i$. Since D is empty, so is D_i, and the claim follows.

Case 2: $D \neq \emptyset$. Recall that for any $v \in D$ there exists i, j such that $j > i+1$ and v is adjacent to r_i and r_j, but not to r_{i+1}. Among all possible $v \in D$ and all possible choices of such i, j for v, choose a v, i, j that minimize $j - i$.

We claim that r_{i+1} is the desired vertex. Note that r_{i+1} is internal: since v is adjacent to r_i and r_j and there is at least one vertex in $V - V_s$ adjacent to r_{i+2} we conclude that there is a path from r_i to r_{i+2} using only vertices in V_s. Also, D_{i+1} must be empty. For assume there exists a $w \in D_{i+1}$, then $w \neq v$ because w is adjacent to r_{i+1} while v is not. Therefore, all neighbours of w must be inside the cycle $\{v, r_i, r_{i+1}, \ldots, r_j, v\}$. Let $j' > i' + 1$ be such that w is incident to $r_{i'}$ and $r_{j'}$, but not to $r_{i'+1}$, then $i' \geq i$ and $j' \leq j$ by the above. In fact, $i' \geq i+1$, because w is incident to r_{i+1}. Therefore $j' - i' < j - i$, contradicting the choice of i and j.

So r_{i+1} is internal and $D_{i+1} = \emptyset$, and we have found the desired vertex.

Thus, as long as G is not completed, we can always apply either Placement I or Placement II and obtain a planar cRI-drawing of a larger subgraph. This proves the following theorem:

Theorem 1. *Any biconnected, chordless, internally triangulated NF3-graph G has a planar closed rectangle of influence drawing.*

3.2 Extension to all NF3-Graphs

Lemma 5. *If G is an NF3-graph, then we can add edges to G until we get a graph that is a biconnected, chordless, internally triangulated NF3-graph.*

Proof. (sketch) We show first that if G has a cutvertex or a chord, then we can always add an edge without creating a filled 3-cycle. Hence we can add edges to G until it is biconnected and chordless and remains an NF3-graph. We can then, similarly to the method shown in [2], add edges to G such that it remains a chordless NF3-graph and becomes internally triangulated.

Theorem 2. *A graph admits a weak planar closed rectangle of influence drawing if and only if it is an NF3-graph.*

Proof. The “only if” direction was shown in Lemma 1. The ”if” direction follows from Theorem 1 and Lemma 5.

Corollary 1. *Any NF3-graph G has a planar open rectangle of influence drawing.*

Proof. The proof follows from Theorem 2 and the fact that a closed rectangle of influence drawing is a weak open rectangle of influence drawing [8].

Corollary 2. *Any 4-connected planar graph that is not triangulated has a planar closed rectangle of influence drawing.*

Proof. This follows by choosing the outer-face of the graph to be the face that is not a 3-cycle; the drawing then has no filled 3-cycles and the graph is an NF3-graph.

4 Conclusion

In this paper, we have studied rectangle of influence drawings, and have shown that a planar graph has a weak planar closed rectangle of influence drawing if and only if it can be embedded without filled 3-cycles. In particular, all 4-connected planar graphs that are not triangulated have such a drawing.

We conclude with remarks and pointers to open problems:

- Some planar graphs (for example K_4) have an open planar rectangle of influence graphs even though they are not an NF3-graph. But other planar graphs (for example the octahedron) can be shown not to have such a drawing. Can we characterize those planar graphs that have a planar open rectangle of influence drawing? Can we find an algorithm to determine whether a planar graph has such a drawing?
- Does every planar graph have a weak open rectangle of influence drawing (it need not be planar)? How about closed rectangle of influence drawings?
- Does every NF3-graph have a strong rectangle of influence drawing?
- In our construction, no two vertices are ever placed in the same row or in the same column. By applying an order-preserving transformation, we thus obtain a drawing in an $n \times n$-grid. But one vertex per row/column seems a waste. Can we show that every NF3-graph has a rectangle of influence drawing of area significantly less than n^2?

References

1. N. Alon, Z. Füredi, and M. Katchalski. Separating pairs of points by standard boxes. *European Journal of Combinatorics*, 6:205–210, 1985.
2. T. Biedl, G. Kant, and M. Kaufmann. On triangulating planar graphs under the four-connectivity constraint. *Algorithmica*, 19(4):427–446, 1997.
3. J. Bondy and U. Murty. *Graph Theory and Applications.* Amerian Elsevier Publishing Co., 1976.
4. M. de Berlin, S. Carlsson, and M. Overmars. A general approach to dominance in the plane. *Journal of Algorithms*, 13:274–296, 1992.
5. G. Di Battista, W. Lenhart, and G. Liotta. Proximinity drawability: A survey. In R. Tamassia and I. Tollis, editors, *DIMACS International Workshop, Graph Drawing 94*, volume 894 of *Lecture Notes in Computer Science*, pages 328–339. Springer-Verlag, 1995.
6. H. de Fraysseix, J. Pach, and R. Pollack. How to draw a planar graph on a grid. *Combinatorica*, 10:41–51, 1990.
7. M. Ichino and J. Sklansky. The relative neighborhood graph for mixed feature variables. *Pattern Recognition*, 18(2):161–167, 1985.
8. G. Liotta, A. Lubiw, H. Meijer, and S. Whitesides. The Rectangle of Influence drawability problem. *Computational Geometry: Theory and Applications*, 10(1):1–22, 1998.
9. M.H. Overmars and D. Wood. On rectangular visibility. *Journal of Algorithms*, 9(3):372–390, 1988.
10. F. Preparata and M. Shamos. *Computational Geometry: An Introduction.* Springer-Verlag, New York, 1985.

Voronoi Drawings of Trees*

Giuseppe Liotta[1] and Henk Meijer[2]

[1] Dipartimento di Ingegneria Elettronica e dell'Informazione,
Università di Perugia, Perugia, Italy.
liotta@diei.unipg.it
[2] Department of Computing and Information Science,
Queen's University, Kingston, Ontario, Canada.
henk@cs.queensu.ca

Abstract. This paper investigates the following problem: Given a tree T, can we find a set of points in the plane such that the Voronoi diagram of this set of points is a drawing of T? We study trees that can be drawn as Voronoi diagrams both in the Euclidean and in the Manhattan metric. Characterizations of drawable trees are given and different drawing algorithms that take into account additional geometric constraints are presented.

1 Introduction

Voronoi diagrams are one of the most studied structures in computational geometry because of their use in key application areas including metrology, astronomy, biology, GIS, computer graphics, and operations research [1,12]. Often papers and book chapters that are devoted to Voronoi diagrams simplify the matter adopting the "no-degeneracies assumption", i.e assuming that no four sites defining the diagram are co-circular (see, e.g., [13]). Under this assumption, Voronoi diagrams in the Euclidean metric have all vertices of degree three. However, the increasing demand of robust and reliable computational geometry methodologies in recent years has been leading to re-studying basic geometric problems and structures within more realistic frameworks in which simplifying assumptions that rule out degenerate configurations of the input are avoided.

This paper studies degenerate Voronoi diagrams, i.e. Voronoi diagrams that contain vertices of degree higher than three originated by four or more co-circular points (also called *sites* in the following). Based on the observation that a typical degeneracy for Voronoi diagrams is the one in which co-circular points give rise to a tree-like structure (see, e.g. [14,12]) we investigate the combinatorial properties of degenerate Voronoi diagrams by asking what type of trees can be the Voronoi diagram of some set of sites in the plane. This question can be naturally turned into the following graph drawing problem: Given a tree T, can one represent T

* Research supported in part by the CNR Project "Geometria Computazionale Robusta con Applicazioni alla Grafica ed al CAD', by the ESPRIT LTR Project 20244 (ALCOM-IT), and by NSERC. The authors wish to thank G. Di Battista for support and useful discussions.

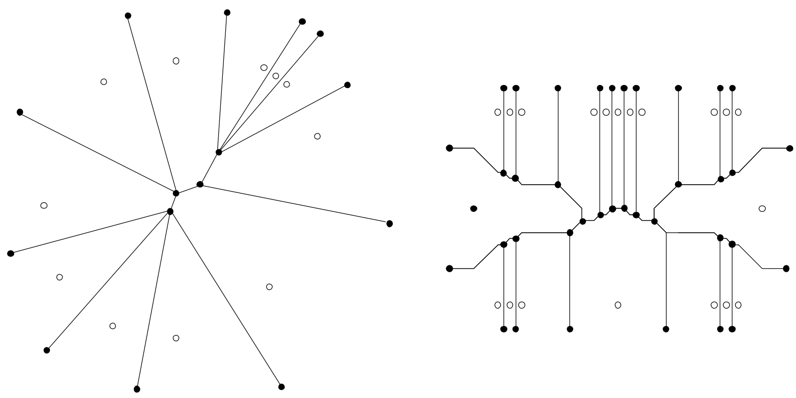
J. Kratochvíl (Ed.): GD'99, LNCS 1731, pp. 369–378, 1999.

so that the resulting drawing is the Voronoi diagram of some set of sites in the plane?

We recall that the the problem of analyzing the combinatorial properties of a given type of geometric graph has a long tradition in the computational geometry and graph drawing communities [2]. We give a few examples below. Monma and Suri [11] proved that each tree with maximum vertex degree at most five can be the Euclidean minimum spanning tree of some set of vertices in the plane, while no tree having at least one vertex with degree greater than six can be drawn as a minimum spanning tree on the plane. As for trees having maximum degree equal to six, Eades and Whitesides [7] showed that it is NP-hard to decide whether such trees can be Euclidean minimum spanning trees on the plane. The problem of representing a tree as a Euclidean minimum spanning tree in the three dimensional space is studied in [8]. Dillencourt has shown that all Delaunay drawable triangulations are 1-tough, and have perfect matchings [5], and that all maximal outerplanar graphs are Delaunay drawable [4]. Dillencourt and Smith [6] show that any triangulation without chords or non facial triangles is Delaunay drawable. The problem of deciding whether a graph can be drawn as rectangle of influence graph and the design of efficient algorithms for computing such a drawing when one exists is investigated in [9]. A survey on the problem of drawing a graph as a given type of geometric graph can be found in [3].

Representing a tree as a Voronoi diagram has many interesting aesthetic features. It captures the natural way of drawing a tree so that groups of adjacent vertices appear close to each other while vertices not incident to a certain edge are drawn far apart from that edge. Also, the shape of the edges varies according to the chosen metric for the Voronoi diagram: in the Euclidean metric the edges follow the *straight-line standard*, i.e. they are straight lines, while in the Manhattan metric the edges follow the *semi-orthogonal standard*, i.e. they are polygonal chains of segments with slope 0, ∞, or ± 1. A drawing of a graph that is the Voronoi diagram of a set of sites when distances are measured according to the Euclidean metric (also called L_2 metric) is a *Euclidean Voronoi drawing*. Similarly, a *Manhattan Voronoi drawing* is a drawing of a graph that is the Voronoi diagram of some set of sites in the plane when distances are measured according to the Manhattan metric (also called L_1 metric). For display purposes, we assume that in a Voronoi drawing all edges have finite length, that is the infinite rays of a Voronoi diagram are replaced by edges of finite length in a Voronoi drawing. A graph that admits a Euclidean (Manhattan) Voronoi drawing is said to be *Euclidean* (*Manhattan*) *Voronoi drawable*. Figure 1 (a) and (b) shows a Euclidean and a Manhattan Voronoi drawing of a tree, respectively. In both figures, the white circles represent the sites whose Voronoi diagram is the drawing of the tree.

It is worth mentioning that the requirement of clustering together adjacent vertices has been already considered in the context of *proximity drawings*, a family of straight-line drawings of graphs which include Gabriel drawings, relative neighborhood drawings, and β-drawings among its representatives. In a proximity drawing two vertices are adjacent if some local proximity constraint



(a) (b)

Fig. 1. (a) Euclidean Voronoi drawing and (b) Manhattan Voronoi drawing of a tree. The white circles are sites, the black circles are vertices of the drawing.

is satisfied; for example, in a Gabriel drawing it is required that the disk having the two vertices as antipodal points does not contain any other vertex. However, the satisfaction of the local proximity requirements in these types of drawings is often at the expenses of other important aesthetic constraints, such as having an uniform edge length and/or having an area of the representation that is polynomial with the size of the input graph (see, e.g. [10]). As a contrast, Euclidean Voronoi drawings only require a screen-area that is polynomial in the input size. For references on proximity drawings of graphs see [3].

The main results of this paper can be summarized as follows.

- We characterize Euclidean Voronoi drawable trees. Such trees are those whose interior vertices have degree at least three. Our result extends a previous result by Dillencourt [4] who proves that all maximal planar graphs are drawable as Delaunay triangulations, thus implying that any tree with internal vertices of degree exactly three is Euclidean Voronoi drawable.
- We present different algorithms that compute Euclidean Voronoi drawings of trees and that can satisfy different additional aesthetic constraints. We show how to compute *radial* Euclidean Voronoi drawings, i.e. Voronoi drawings of rooted trees in which the root of the tree is placed at the center of a disk and all vertices at distance k from the root are drawn on a disk of radius k and having center at the root. We also show how to compute *uni-length* Euclidean Voronoi drawings of trees, i.e. Euclidean Voronoi drawings where all edges have the same length. Further, we show that the above drawing can be computed so that all the the angles between the outgoing edges of each vertex in the drawing are constrained to be equal. The time complexity of the proposed algorithms is linear in the real RAM model of computation.

- Motivated by the graphic appeal of drawing trees in a semi-orthogonal standard, we extend the investigation from the Euclidean to the Manhattan metric and study which trees are Manhattan Voronoi drawable. While the study of the Euclidean Voronoi drawable trees relies on a well-know theorem (see, e.g., [13]) which characterizes point sets whose Voronoi cells in the Euclidean metric are unbounded, as far as we know there is not a counterpart of such theorem in the Manhattan metric. We start our investigation of Manhattan drawings of trees by giving a characterization of the sets of points whose Voronoi diagrams in the Manhattan metric are trees.
- Since in the semi-orthogonal standard there cannot be more than eight edges incident on the same vertex there is a natural upper bound on the vertex degree of Manhattan Voronoi drawable graphs. Surprisingly, not only there cannot be Voronoi drawable trees whose vertices have degree eight, but we also show that there cannot exist drawable trees with vertices having degree larger than five. We also give examples of Manhattan drawable trees having vertices of degree five.
- We characterize those binary trees that are Manhattan Voronoi drawable. This result can be seen as the counterpart for the Manhattan metric of the result by Dillencourt [4] for the Euclidean metric.

For reasons of space, some proofs have been omitted or sketched in this extended abstract.

2 Preliminaries

Let $P = \{p_0, p_1, \ldots, p_{n-1}\}$ denote a set of n distinct points in the plane. A point $p \in P$ will be called a *site*. For a given metric, we denote with $d(u, v)$ the distance function between two points u and v.

Under the Euclidean or L_2 metric, the *Voronoi region* of a site p_i, denoted by $V(p_i)$ is the set of all points q for which $d(q, p_i) \leq d(q, p_j)$ for all sites p_j, with $i \neq j$. The set of all points q for which $d(q, p_i) = d(q, p_j)$ is the *Euclidean separator* of p_i and p_j and is denoted as $Sep(p_i, p_j)$. Under the Manhattan or L_1 metric, this definition is as follows [1]. Assume that p_i is lexicographically smaller than p_j and let $Dom(p_i, p_j) = \{q | d(q, p_i) < d(q, p_j)\}$ and $Dom(p_j, p_i) = \{q | d(q, p_j) \leq d(q, p_i)\}$. The *Manhattan separator* of sites p_i and p_j, denoted by $Sep(p_i, p_j)$, is defined as the common boundary of $Dom(p_i, p_j)$ and $Dom(p_j, p_i)$. The Voronoi region of a site p_i under the L_1 metric is $V(p_i) = \bigcap_{j \neq i} Dom(p_i, p_j)$. Therefore the Voronoi region of a site in the L_1 metric in general does not contain all points of its boundary and may contain areas of thickness 0. Since the characterization results and the drawing techniques of this paper are not affected by whether the Voronoi cells contain their boundaries or have areas of thickness 0, we assume throughout the paper that a Voronoi region of a site always contains its boundary both in the Euclidean and in the Manhattan metric, and does not contain areas of thickness 0.

The *Voronoi diagram* of P, denoted by $Vor(P)$ is the collection of all boundaries of the Voronoi regions $V(p_i)$. A point that lies in at least three different

Voronoi regions is called a *Voronoi vertex*. Since vertices of a Voronoi diagram lie in at least three Voronoi regions, Voronoi drawable trees do not have vertices of degree 2.

Let p be a point in the plane and let l_- and l_+ be two lines of slope -1 and $+1$ through p respectively. The lines l_- and l_+ define four convex cones with apex p. We call *East cone of* p, denoted by $EC(p)$, the cone to the right of p, i.e. the cone that contains the positive $x-axis$ with p as origin. Similarly, we define the *North cone of* p ($NC(p)$), *West cone of* p ($WC(p)$), and *South Cone of* p ($SC(p)$) as the cones with apex p and that are above, to the left, and below p respectively. The cones $EC(p)$, $NC(p)$, $WC(p)$ and $SC(p)$ do not include points of l_- or l_+. The set of points of l_+ with $x-$ and $y-$coordinates larger than those of p is called the *north-east bisector of* p and is denoted by $ne(p)$. The *north-west bisector of* p, denoted as $nw(p)$ is the portion of l_- whose points have larger y-and smaller $x-$coordinates than p does. Similarly, *the south-east bisector of* p (denoted as $se(p)$), and the *south-west bisector of* p (denoted as $sw(p)$) can be defined as the portions of l_+ and l_- that lie in the third and fourth quadrant of p and that do not contain p.

Let $C(p)$ be one of the four cones of p defined above, i.e. $C = EC$, NC, WC or SC. We use the notation $C[p]$ to denote the cone $C(p)$ including its boundary (but excluding the apex p). $C[p)$ includes only the right boundary when the interior of the cone is viewed from p, and $C(p]$ is the cone plus the left boundary. For example, $EC[p] = EC(p) \cup se(p) \cup ne(p)$, $NC[p) = NC(p) \cup ne(p)$ and $NC(p] = NC(p) \cup nw(p)$. Finally we denote by $e(p)$ and $w(p)$ the points on the horizontal line through p to the right and left of p respectively, and by $n(p)$ and $s(p)$ the points on the vertical line through p above and below p respectively.

3 Euclidean Voronoi Drawings of Trees

In this section we first characterize the family of Euclidean Voronoi drawable trees and then show different algorithms that compute Euclidean Voronoi drawings of trees. Our characterization is based on the following result.

Theorem 1. **[13]** *Let* P *be a set of sites and let* $p \in P$. *The Voronoi region of* p *is unbounded if and only if* p *is a vertex of the convex hull of* P.

By Theorem 1, the Voronoi diagram of P is a tree if and only if all sites of P are in convex position. In the next lemma we study sets of sites in convex position.

Lemma 1. *Each tree without vertices of degree two is Euclidean Voronoi drawable.*

Proof. Let T be a tree without vertices of degree two. We prove the lemma by defining a set of sites P, for which $Vor(P)$ is isomorphic to T. The edges of $Vor(P)$ that have infinite length are replaced by edges of finite length at the end of the construction. We root T at an arbitrary non-leaf vertex r. Suppose r

has degree k. Place a point c_0 on the plane at an arbitrary location and draw a circle C_0 around c_0 with radius 1. Place k sites on C_0 at arbitrary locations. The Voronoi diagram of these sites is a tree with a single non-leaf vertex of degree k placed at c_0; c_0 is the Voronoi vertex that represents r. For each site around C_0, draw a line tangent to C_0 through the site. We say that a point lies on the inside of one of these tangent lines if it lies on the same side as c_0. The remaining sites of P that define the other internal Voronoi vertices of T will all be placed outside the circle C_0 but inside the tangent lines.

We define the other points of P such that $Vor(P)$ is isomorphic to T by traversing T from r to the leaves. The vertices at depth m (i.e. the vertices whose path from r consists of m edges) are drawn only after all vertices at depth $m-1$ have been drawn. Suppose we want to draw vertex v_j whose parent v_i has been already drawn and let c_i be the Voronoi vertex representing v_i. Vertex v_j is drawn by defining a new disk whose center is on one of the infinite Voronoi edges incident on c_i. Let $e_j = Sep(p, q)$ be this edge, where p and q are two sites already defined in a previous step of the algorithm. Let C be the set of all circles whose centers define the Voronoi vertices drawn so far. Let L be the set of all lines that are tangent to the circles in C and that pass through the sites that have already been placed.

The following tasks are executed.

- Place vertex c_j on the edge e_j, and draw the circle C_j with center c_j through p and q. Place c_j sufficiently close to c_i so that C_j is partially inside all tangent lines in L and outside all circles in C.
- Let k_j be the degree of vertex v_j in T. Place $k_j - 1$ sites on the portion of C_j that lies inside all tangent lines in L and outside all circles in C.
- Draw the $k_j - 1$ lines through the new sites tangent to C_j. Add these lines to L and add C_j to C.

We can derive that the algorithm is correct based on the following two observations. Firstly, we can always place c_j close enough to c_i so that C_j falls partially inside all tangent lines and outside all circles. Secondly the $k - 1$ sites we place on C_j are closer to c_j than to any other vertex c_i, so the vertices of the Voronoi diagram of previously placed sites do not change.

The drawing algorithm defined in the proof of Lemma 1 can be modified in order to satisfy additional geometric constraints that define *aesthetic criteria* traditionally used in graph drawing to improve the readability of the drawings. For the definition of radial drawing and uni-length drawing see [2].

Theorem 2. *Any tree T whose internal vertices have at least three incident edges is Euclidean Voronoi drawable. A Euclidean Voronoi drawing Γ of T can be computed in time proportional to the size of T in the real RAM model of computation. Γ can satisfy additional aesthetic requirements:*

- *Γ can be a radial drawing, such that for each vertex $v \in \Gamma$ all outgoing angles of v are the same.*
- *Γ can be a uni-length drawing, such that for each vertex $v \in \Gamma$ all outgoing angles of v are the same.*

4 Point Sets in the Manhattan Metric

We start our investigation of Manhattan Voronoi drawings of trees by giving a characterization of the sets of points whose Voronoi diagrams in the Manhattan metric are trees. This is achieved by independently characterizing sets of points whose Manhattan Voronoi diagrams are acyclic and those whose Manhattan Voronoi diagrams are connected.

Under the L_1 metric, a Manhattan separator $Sep(p_i, p_j)$ is a semi-orthogonal polygonal chain, i.e. a chain whose segments can have slope 0, ∞, and ± 1. An edge of a Manhattan Voronoi drawing is a subset of a Manhattan separator and hence it can have bends, each bend defining a turn of an angle that is a multiple of $\pi/4$. The next property studies what a Manhattan separator can look like for a set of two sites.

Property 1. Let p and q be two sites. The eight possible shapes of the Manhattan separator $Sep(p, q)$ are shown in figure 2 (a).

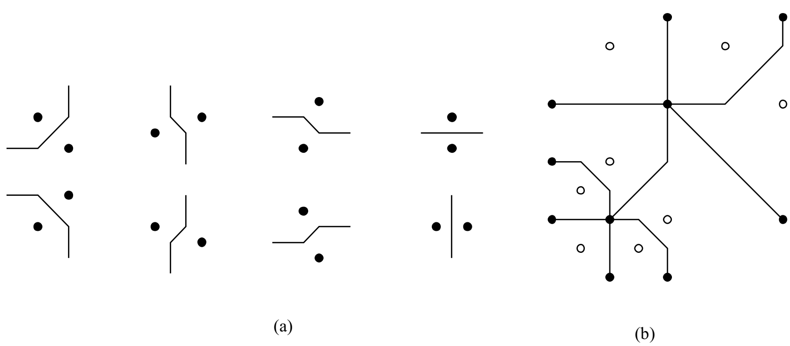


Fig. 2. (a) Manhattan separators of two sites. (b) Manhattan Voronoi drawing of a tree with two degree vertices of degree five.

We say that p *has the empty East cone property* if $EC[p]$ does not contain sites. Similarly, p *has the empty North*, *empty West*, and *empty South cone property* if $NC[p)$ does not contain sites, $WC[p)$ or $WC(p]$ does not contain sites, and $SC(p]$ does not contain sites respectively. We say that p *has an empty cone property* if it has either the empty East, or North, or West, or South cone property. By exploiting Property 1 and the fact that the Voronoi region $V(p)$ of a site p is a star-shaped polygon with p in its kernel, the following can be proved.

Theorem 3. *A Voronoi diagram Vor(P) is acyclic if and only if all sites p have an empty cone property.*

The Voronoi diagram of a set of sites in the Manhattan metric may be not connected. Intuitively, a Voronoi diagram is not connected when there exists a

site such that two of its four cones are empty while the other two contain sites. However, since the cones may or may not contain their boundaries and since there can be sites aligned along lines with slope ± 1, a more precise statement of this condition is needed.

Lemma 2. *If there is a site $p \in P$ such that $EC[p]$ does not contain sites, $NC(p]$ contains sites, $WC(p]$ or $WC[p)$ does not contain sites, and $SC[p)$ contains sites, then $Vor(P)$ is not connected.*

Lemma 3. *If there is a site $p \in P$ such that $EC(p)$ contains sites, $NC[p)$ does not contain sites, $WC[p]$ contains sites, and $SC(p]$ does not contain sites, then $Vor(P)$ is not connected.*

Lemma 4. *If there is a site $p \in P$ such that $EC(p] \cup NC[p)$ contains sites, $WC[p) \cup SC(p]$ does not contain sites, and $sw(p)$ contains sites, then $Vor(P)$ is not connected.*

Lemma 5. *If there is a site $p \in P$ such that $SC(p] \cup EC[p)$ contains sites, $NC[p) \cup WC(p]$ does not contain sites, and $nw(p)$ contains sites, then $Vor(P)$ is not connected.*

A site p is called a *separating site* if at least one of the sets of the conditions in the Lemmas 2, 3, 4, or 5 holds.

Theorem 4. *Let P be a set of sites and let $Vor(P)$ be the Voronoi diagram of P in the L_1 metric. $Vor(P)$ is connected if and only if there exists no separating site $p \in P$.*

Theorem 5. *Let P be a set of sites and let $Vor(P)$ be the Voronoi diagram of P in the L_1 metric. $Vor(P)$ is a tree if and only if there is no separating site in P and each site $p \in P$ has an empty cone property.*

5 Manhattan Voronoi Drawings of Trees

Because of Property 1, the number of distinct edges incident on a same vertex of a Manhattan Voronoi drawing of a graph is at most eight. Hence, the class of Manhattan drawable trees is a contained in the class of those trees having maximum vertex degree at most eight. However, there is a smaller upper bound on the degree that the vertices of a Manhattan Voronoi drawable tree can have. Namely, by analyzing the geometry of the edges incident on a same Voronoi vertex it can be shown that there cannot be six or more edges incident on the same vertex. Based on Figure 2(b) that shows an example of a Manhattan Voronoi drawing of a tree with vertices of degree five, we can conclude the following.

Theorem 6. *A Manhattan Voronoi drawable tree cannot have vertices with six or more incident edges. Also, there exist Manhattan Voronoi drawable trees that have vertices with five incident edges.*

Not all trees having vertices of degree at most five are Manhattan Voronoi drawable. To prove this claim, we characterize those binary trees that are Manhattan Voronoi drawable. We start by introducing some terminology.

Let T be binary a tree without vertices of degree two. T is a *caterpillar* if each vertex of T is either a leaf or the neighbour of a leaf. Hence, a caterpillar contains a path that connects two leaves plus an arbitrary number of side branches consisting of single edges. The path is called *backbone* of the caterpillar; the branches are called *legs* of the caterpillar. Each branch shares a vertex with the backbone. If the caterpillar is a single edge, then it coincides with its backbone and there are no branches.

The class of binary trees that we are going to study consists of caterpillars "joined" to some "skeleton", which we define as follows. Let T be binary tree without vertices of degree two. A *spine* of T is a subtree of T containing at least two and at most most four leaves of T and containing all vertices and edges of the paths between these leaves. We say that a *caterpillar is attached to a spine* if one of the endpoints of its backbone is a non-leaf vertex of the spine.

Lemma 6. *Any binary tree T that has no vertices of degree 2 and that consists of an arbitrary number of caterpillars attached to a spine sp is Manhattan Voronoi drawable.*

Proof. We prove the lemma assuming that that *sp* has four leaves. If the spine *sp* has fewer that 4 leaves, a similar reasoning can be used.

Notice that *sp* has two vertices v_0 and v_1 of degree 3 whose neighbours are in *sp*. We explain how a set of sites P can be constructed such that the Voronoi diagram of P in the Manhattan metric is isomorphic to T.

We distinguish two different cases. Case (a): if there are no vertices between v_0 and v_1 then we start the construction as follows: sites p_0 p_1, p_2 and p_3 are placed at points (0,4), (6,0), (0,-4) and (-6,0). Case (b): if there are vertices between v_0 and v_1 then we start the construction as follows: sites p_0 p_1, p_2, p_3 and p_4 are placed at points (-1,4), (1,4), (6,0), (0,-4) and (-6,0).

We now add sites such that the Voronoi diagram of these sites includes the backbones of all caterpillars, but not yet the legs. This can be done by adding sites on the lines $y = 4$ and $y = -4$: place an arbitrary number of sites at points with $x-$ coordinates between -5 and -2 and between 2 and 5. For case (b), we can also place sites on the line $y = 4$ with $x-$ coordinates between -1 and 1. It can be verified that the Voronoi vertices of the sites p_0, p_1, p_2, p_3 and p_4 do not move (except for the Voronoi vertex of p_0, p_1 and p_2), so the backbones of the caterpillars are attached to the correct sections of the spine.

Finally, we add legs to the backbones of the caterpillars as follows. Suppose that between sites q_0 and q_1 there is a caterpillar with k legs. Assume that q_1 is to the right of q_0. Place k sites on the ray $nw(q_1)$, close enough to q_1 to ensure that $NC[q_0)$ remains empty. Notice that the new sites do not change the location of any of the existing vertices in the Voronoi diagram of P, except the location of the vertex corresponding to p_0, p_1 and p_3 in case (b). It can be seen that the new sites create the legs as required. Therefore, all trees that can be decomposed into a spine and a set of caterpillars are Manhattan Voronoi drawable.

Let P be a set of sites whose Voronoi diagram $Vor(P)$ is a tree in the Manhattan metric. The proof of the following lemma is based on an analysis of the properties of the cones defined by the sites in P.

Lemma 7. *Any binary tree that is the Voronoi diagram of a set of sites under the L_1 metric is a tree T that has no vertices of degree two and consists of an arbitrary number of caterpillars attached to a spine.*

Lemmas 6 and 7 prove the following.

Theorem 7. *A binary tree T is Manhattan Voronoi drawable if and only if it has no vertices of degree 2 and consists of an arbitrary number of caterpillars attached to a spine.*

References

1. F. Aurenhammer. Voronoi diagrams: A survey of a fundamental geometric data stru cture. *ACM Comput. Surv.*, 23(3):345–405, Sept. 1991.
2. G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. *Graph Drawing.* Prentice Hall, Upper Saddle River, NJ, 1999.
3. G. Di Battista, W. Lenhart, and G. Liotta. Proximity drawability: a survey. In R. Tamassia and I. G. Tollis, editors, *Graph Drawing (Proc. GD '94)*, volume 894 of *Lecture Notes Comput. Sci.*, pages 328–339. Springer-Verlag, 1995.
4. M. B. Dillencourt. Realizability of Delaunay triangulations. *Inform. Process. Lett.*, 33(6):283–287, Feb. 1990.
5. M. B. Dillencourt. Toughness and Delaunay triangulations. *Discrete Comput. Geom.*, 5:575–601, 1990.
6. M. B. Dillencourt and W. D. Smith. Graph-theoretical conditions for inscribability and Delaunayr ealizability. In *Proc. 6th Canad. Conf. Comput. Geom.*, pages 287–292, 1994.
7. P. Eades and S. Whitesides. The realization problem for Euclidean minimum spanning trees is NP-hard. *Algorithmica*, 16:60–82, 1996. (special issue on Graph Drawing, edited by G. Di Battista and R. Tamassia).
8. G. Liotta and G. Di Battista. Computing proximity drawings of trees in the 3-dimensional space. In *Proc. 4th Workshop Algorithms Data Struct.*, volume 955 of *Lecture Notes Comput. Sci.*, pages 239–250. Springer-Verlag, 1995.
9. G. Liotta, A. Lubiw, H. Meijer, and S. H. Whitesides. The rectangle of influence drawability problem. *Comput. Geom. Theory Appl.*, 10(1):1–22, 1998.
10. G. Liotta, R. Tamassia, I. G. Tollis, and P. Vocca. Area requirement of Gabriel drawings. In *Algorithms and Complexity (Proc. CIAC' 97)*, volume 1203 of *Lecture Notes Comput. Sci.*, pages 135–146. Springer-Verlag, 1997.
11. C. Monma and S. Suri. Transitions in geometric minimum spanning trees. *Discrete Comput. Geom.*, 8:265–293, 1992.
12. A. Okabe, B. Boots, and K. Sugihara. *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams.* John Wiley & Sons, Chichester, UK, 1992.
13. F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction.* Springer-Verlag, 3rd edition, 1988.
14. K. Sugihara and M. Iri. Construction of the Voronoi diagram for 'one million' generat ors in single-precision arithmetic. *Proc. IEEE*, 80(9):1471–1484, Sept. 1992.

Infinite Trees and the Future*

Extended Abstract

Camil Demetrescu[1], Giuseppe Di Battista[2], Irene Finocchi[3], Giuseppe Liotta[4], Maurizio Patrignani[2], and Maurizio Pizzonia[2]

[1] Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza",via Salaria 113, 00198 Roma, Italy.
demetres@dis.uniroma1.it

[2] Dipartimento di Informatica e Automazione, Università di Roma Tre,via della Vasca Navale 79, 00146 Roma, Italy.
{gdb,patrigna,pizzonia}@dia.uniroma3.it

[3] Dipartimento di Scienze dell'Informazione, Università di Roma "La Sapienza", via Salaria 113, 00198 Roma, Italy.
finocchi@dsi.uniroma1.it

[4] Dipartimento di Ingegneria Elettronica e dell'Informazione, Università di Perugia,via G. Duranti 93, 06125 Perugia, Italy.
liotta@diei.unipg.it

Abstract. We study the problem of designing layout facilities for the navigation of an "infinite" graph, i.e. a graph that is so large that its visualization is unfeasible, even by gluing together all the screen snapshots that a user can take during the navigation. We propose a framework for designing layout facilities that support the navigation of an infinite tree. The framework allows to exploit the knowledge of future moves of the user in order to reduce the changes in her mental map during the navigation. Variants of the classical Reingold-Tilford algorithm are presented and their performance is studied both experimentally and analytically.

1 Introduction

Designing layout facilities for the navigation of "infinite" graphs is one of the challenges of the Graph Drawing field. By "infinite" we mean that it would be unfeasible to visualize the graph entirely, even by gluing together all the screen snapshots that a user can take during the navigation. Examples of graphs that can be considered infinite in the above sense come from disparate application areas including World Wide Web, Knowledge Bases, Communication Networks, and Semantic Networks. Several papers on the visualization of very large graphs appear in the literature. A limited list includes [9,13,11,20,16,17,6]. However, the classical assumption is that the input graph, even if huge, is completely known in advance.

* Research supported in part by the CNR Project "Geometria Computazionale Robusta con Applicazioni alla Grafica ed al CAD", and by the ESPRIT LTR Project 20244 (ALCOM-IT)

J. Kratochvíl (Ed.): GD'99, LNCS 1731, pp. 379–391, 1999.

We adopt a different point of view making the assumption, similar to the one in [7], that either the graph is too large to be entirely known or it is practically unfeasible to draw it all. Hence, during the navigation, the user is allowed to see only the content of a limited size window that she can move for exploring the graph. It is clear that such a window cannot be purely geometric: If the graph is so large to be impossible to visualize or even to know, then the idea of moving a window on a pre-computed drawing does not make sense. Therefore, our idea of window is that of a topological window, defined in terms of the structure of the graph. For example, at each time of the navigation the topological window may display the subgraph induced by the vertices that have a constant topological distance from a given vertex, considered as the center of the window.

Hence, the graph is displayed as a sequence of drawings determined by the navigation path followed by the user. Since consecutive drawings in the sequence can have a large overlap, it is essential to devise visualization strategies that preserve the mental map [8,14,12,2] of the user. Algorithms devised to preserve the mental map are usually evaluated in terms of both static and dynamic quality measures [15,3]. Static measures evaluate standard Graph Drawing aesthetics like number of crossings or area used by the drawing. Dynamic measures evaluate how much a drawing is similar to the one(s) preceding it in the sequence.

Our work starts from two main observations: (1) While the display area is bounded by the size of the screen, the size of the portion of graph that the system can know at each instant of time is only bounded by the resources of the computer. (2) Very often, even if the graph is infinite, the visit of vertices follows an almost predictable pattern. For example, it can be experimentally verified that most users access a Web site following a limited set of favorite patterns. Thus, the following question naturally rises. Is it possible to devise a drawing strategy that takes advantage of the knowledge of part of the future navigation moves of the user in order to achieve better performance in terms of dynamic quality measures? In other words, suppose that the user is observing the drawing of a certain subgraph G_1 and suppose to know in advance that the next navigation step will require the visualization of a subgraph G_2; given this knowledge, is it possible to draw G_1 so to reduce the changes in the overlapping portions of the drawings of G_1 and G_2? The idea is to exploit the knowledge of future moves in the current visualization. It is interesting to note that the importance of knowing in advance the sequence of graphs to display within an off-line visualization framework has been discussed in [18].

The main components of our framework are: A *visualization window* defining, at each step of the navigation, what is the graph (a portion of the infinite graph) to visualize. A *knowledge window* defining, at each step of the navigation, a supergraph of the visualized one. The size of the knowledge window affects the drawing of the visualization window. A *favorite neighbor* for each vertex of the infinite graph. The favorite neighbor defines what is the next navigation step that the user is going to perform.

As an application of the above framework, we study the problem of navigating in an infinite tree. In spite of the structural simplicity of trees, designing drawing

strategies for navigating in infinite trees offers several experimental and theoretical challenges. Also, the Web is often visualized as a tree-like structure (see e.g. [1,10]). We devise variants of the classical Reingold-Tilford algorithm [19]. This choice is motivated by the fact that the Reingold-Tilford algorithm is among the most robust, simple, effective, and well known algorithms of Graph Drawing. The main results of this paper can be listed as follows. We formally define a framework for designing Graph Drawing facilities that support the navigation of an infinite graph (Section 2). The framework is especially targeted to take into account the knowledge of the future moves of the user. We design a set of simple strategies based on the Reingold-Tilford algorithm for visualizing infinite trees (Section 2). We perform an experimental analysis of the above strategies (Section 3). The analysis shows that exploiting the knowledge of the future moves of the user is not a trivial task, and that some simple strategies can have more drawbacks than benefits from the knowledge of the future. Motivated by the experimental results, we perform a theoretical analysis of the drawing algorithms (Section 4). The analysis both explains our experimental results and allows us to compare the performance of the drawing algorithms as the size of the knowledge window goes to infinity.

Through the paper we make use of standard Graph Drawing terminology [5]. For reasons of space, some proofs are only sketched or omitted in this extended abstract.

2 The Framework and the Drawing Strategies

All our trees are finite subtrees of an infinite tree T and each edge is oriented from the parent to the child. The time of the framework is discrete. Hence, it assumes integer values starting from 0. At instant t the user can look at a *visualization window* of a given positive integer *size* h. The visualization window is a subtree $T_h(t)$ of T rooted at a certain vertex $r(t)$ and induced by the vertices of T at (oriented) distance at most h from $r(t)$. Vertex $r(t)$ is the *point of view* of the user at instant t. If the point of view at instant t is $r(t)$, then the user can move it at instant $t+1$ to any child of $r(t)$. A *navigation* of T in any interval $[t_1, t_2]$ of time consists of the path (*visualization path*) $r(t_1), \ldots, r(t_2)$. The overlap between two visualization windows at consecutive instants t and $t+1$ is $T_h(t) \cap T_h(t+1) = T_{h-1}(t+1)$. An overlap between consecutive visualization windows is illustrated in Figure 1.

From the point of view of the readability of the visualization, we consider static and dynamic quality measures. The static quality measures evaluate the drawings of $T_h(t)$ and of $T_h(t+1)$ separately. Among the possible static quality measures we focus on the width of the drawing, since it is one of the few degrees of freedom when drawing a rooted tree. The dynamic quality measures evaluate the variations in the overlap of the drawings of $T_h(t)$ and $T_h(t+1)$. Namely, they measure the differences between the drawing of $T_{h-1}(t+1)$ inside the drawing of $T_h(t)$ and the drawing of $T_{h-1}(t+1)$ inside the drawing of $T_h(t+1)$. In order to optimize the drawing with respect to the dynamic quality measure we

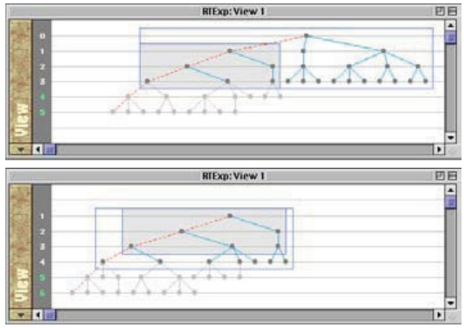


Fig. 1. Two successive snapshots produced by the system Leonardo [4] showing the effect of a single navigation step in a randomly generated tree: the light boxes show $T_h(t)$ and $T_h(t+1)$, the shaded boxes show the portion of the tree visible both before and after the navigation step, that is $T_{h-1}(t+1)$. The value of h is 3.

exploit the following two observations: (1) Even if the visualization window is constrained to have size h, the drawing algorithm that computes the drawing of the visualization window at instant t can know a subtree of T larger than $T_h(t)$. This subtree is denoted as $T_{h+k}(t)$. It has root $r(t)$ and is induced by the vertices of T at (oriented) distance at most $h+k$ from $r(t)$. Constant $k \geq 0$ is an integer that describes how much the algorithm is allowed to know about T more than what belongs to the visualization window. Tree $T_{h+k}(t)$ is called *knowledge window*; $h+k$ is the *size* of the knowledge window. (2) Very often, even if the graph is infinite, the visit of vertices follows an almost predictable pattern. One possibility for modeling this issue is to weight the children of each vertex with their probability to be visited after their parent. We make here the simpler assumption that for each vertex one *favorite child* is defined. Under this assumption the visualization path in any interval $[t_1, t_2]$ is $r(t_1), \ldots, r(t_2)$, where $r(t+1)$ $(t_1 \leq t < t_2)$ is always the favorite child of $r(t)$.

The drawing algorithms that we study are variants of the classical Reingold-Tilford algorithm [19]. This choice is motivated by the fact that the Reingold-Tilford algorithm is among the most robust, simple, effective, and well known algorithms of Graph Drawing.

Suppose to have a visualization window of size h and to have a knowledge window of size $h+k$. At instant t our algorithms compute a drawing of a tree larger than $T_h(t)$. Namely, they receive as input $T_{h+k}(t)$ and perform as follows. They prune $T_{h+k}(t)$ of the vertices that will never be part of any visualization window during the navigation. The resulting pruned tree is called $T_{h,k}(t)$ and is defined as $T_{h,k}(t) = \bigcup_{i=0}^{k} T_h(t+i)$. Then, they compute a drawing of $T_{h,k}(t)$ and display the portion $T_h(t)$ of height h.

Note that the definition of tree $T_{h,k}(t)$ relies on the knowledge of the vertices $r(t), r(t+1), \ldots, r(t+k)$ of the visualization path. Also, observe that $T_{h,k}(t)$ has height $h+k$ and that $T_{h,0}(t) = T_h(t)$. Figure 2 shows a portion of an infinite tree

with a visualization window of height $h = 3$ and a knowledge window of height $h + k = 9$. In the figure, the black vertices belong to the visualization path and together with the grey vertices induce $T_{h,k}(t)$.

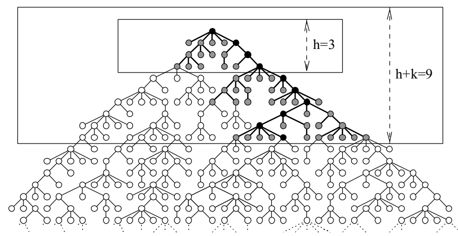


Fig. 2. A portion of an infinite tree, a visualization window of height $h = 3$, and a knowledge window of height $h + k = 9$. A drawing algorithm computes a drawing of $T_{h,k}(t)$ (grey and black vertices). Only the portion of the drawing inside the visualization window is displayed on the screen. The portion of the tree induced by the white vertices is not taken into account by the drawing algorithm.

Our variants of the Reingold-Tilford Algorithms are characterized by a different choice for the embedding of $T_{h,k}(t)$. `Leftmost-Embedding Algorithm`: The embedding of $T_{h,k}(t)$ is chosen so that for each vertex v along the visualization path, the favorite child of v is the leftmost child of v. As a result, the leftmost vertex at any given level of $T_{h,k}(t)$ is a vertex of the visualization path. `Central-Embedding Algorithm`: The embedding of $T_{h,k}(t)$ is chosen so that for each vertex v along the visualization path, the favorite child of v is the middle child of v. A vertex u is the middle child of v if the number of vertices preceding u in the adjacency list of v differs by at most one from the number of vertices following u in the adjacency list of v. `Random-Embedding Algorithm`: The embedding of $T_{h,k}(t)$ is chosen so that the favorite child of each vertex v along the visualization path can be in any position of the adjacency list of v.

We are interested in evaluating the performance of the above algorithms. The *total-shift* dynamic quality measures the change of the x-value of a non-root vertex v of $T_{h-1}(t+1)$ with respect to its parent $p(v)$. Since our algorithms perform a pruning step before computing the drawing, we evaluate the total-shift dynamic quality measure with respect to pruned subtrees. Namely, we denote the total-shift dynamic quality measure as $m_D(h,k,t)$ and define it as follows: $m_D(h,k,t) = \sum_{v \in T_{h-1}(t+1) - \{r(t+1)\}} |x(v,t) - x(p(v),t) - (x(v,t+1) - x(p(v),t+1))|$, where $x(v,t)$ is the x-coordinate of vertex v at instant t computed by an algorithm that considers $T_{h,k}(t)$. The static quality measure that we use evaluates the width of the drawing of the visualization window. More precisely, let $B_{h,k}(t)$ be the smallest isothetic box that contains the first h levels of a drawing of $T_{h,k}(t)$. We denote as $m_S(h,k,t)$ the width of box $B_{h,k}(t)$.

3 Experimental Results

This section contains the results of an experimental comparison of `Random-Embedding Algorithm`, `Central-Embedding Algorithm`, and `Leftmost-Embedding Algorithm` with respect to the dynamic and static quality measures m_D and m_S introduced in Section 2. In each experiment we consider a random path on a randomly generated tree. Since we actually perform a finite number l of navigation steps on each tree, once the visualization window height h and the knowledge k have been fixed, it suffices producing a tree with height $h+k+l$ to simulate the behavior of the algorithms on an infinite tree. Furthermore, in order to simulate a random infinite path on an infinite tree, we need to avoid choosing those random paths ending with a leaf before the expected length is reached.

The generation of a random tree and of the corresponding random path is performed as follows. In order to generate a tree of given height and with vertex outdegree ranging in a fixed interval, we randomly choose in such interval the number of children of each vertex, according to the uniform distribution, starting from the root and considering one level at a time, until we reach the desired tree height. Then we randomly select a leaf on the last level of the tree, implicitly defining a random path from the root to the chosen leaf. In each experiment we fix the parameters h, k, and l, we randomly generate a tree of height $h+k+l$ and a path over it, and we perform l navigation steps, computing at each step the value of $m_D(h,k,t)$ and $m_S(h,k,t)$. For each tree, we average the values of the dynamic and static quality measures over the whole navigation and then we average these values over the entire set of randomly generated trees. The resulting measures are denoted as $m_D(h,k)$ for the total-shift dynamic quality measure and $m_S(h,k)$ for width static quality measure. Instead of plotting the raw values of $m_D(h,k)$ and $m_S(h,k)$, we display their relative improvements, denoted as $i_D(h,k) = \frac{m_D(h,0)-m_D(h,k)}{m_D(h,0)}$ and $i_S(h,k) = \frac{m_S(h,0)}{m_S(h,k)}$, respectively.

Figure 3 shows the results of the experiments performed with $h=5$, $0 \le k \le 6$, minimum vertex outdegree 0, and maximum vertex outdegree between 2 and 6. In particular, Figure 3.a, Figure 3.c, and Figure 3.e show the values of $i_D(h,k)$ for the `Random`, `Central`, and `Leftmost-Embedding Algorithm`, respectively, while Figure 3.b, Figure 3.d, and Figure 3.f show the corresponding values of $i_S(h,k)$. Each point on the diagrams is obtained computing the average over 1000 trees. The axes scales are chosen with the purpose of producing diagrams in the range $[0,1] \times [0,1]$. Concerning the static quality measure, the behavior of $i_S(h,k)$ shows the expected worsening of the width of the drawing k increases. For what concerns the dynamic quality measure, the diagrams show that the best values are obtained by `Leftmost-Embedding Algorithm` and the worst values by `Central-Embedding Algorithm`. In all charts, as the average outdegree of tree T increases, the dynamic improvement for any fixed value of k decreases. Also notice that $i_D(h,k)$ measured for `Random` and `Central-Embedding Algorithm` can be negative when the outdegree of the vertices increases (see Figure 3.a and Figure 3.b). This shows that not all strategies are able to take advantage of the knowledge of the future. On the other hand, $i_D(h,k)$ is always positive for

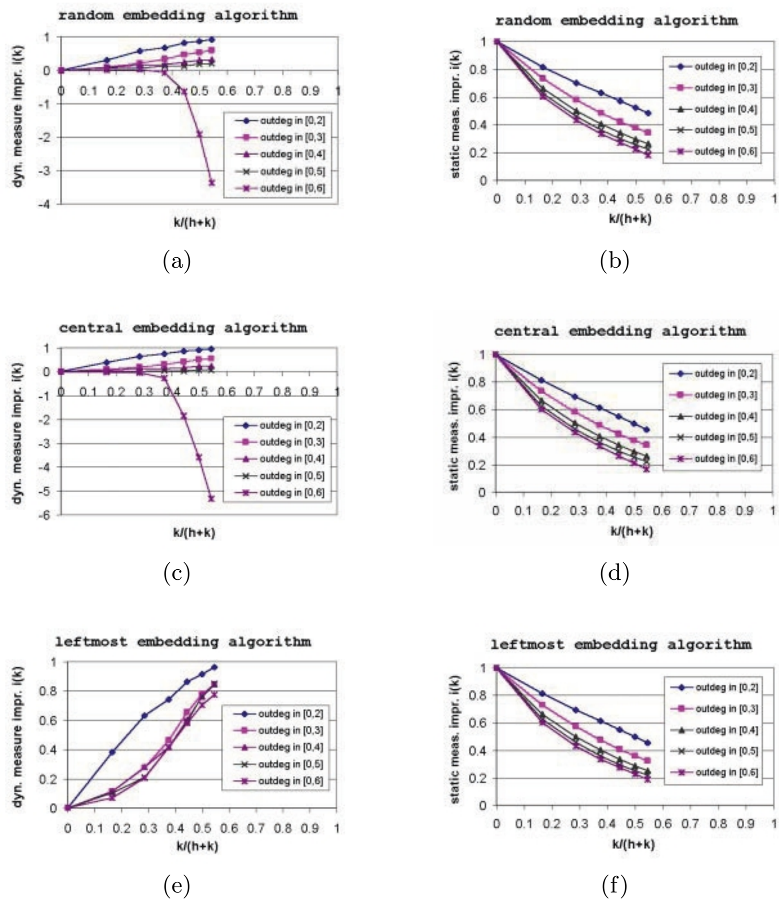


Fig. 3. Experimental results for $h = 5$, $0 \leq k \leq 6$, minimum vertex outdegree 0, and maximum vertex outdegree between 2 and 5. On the left (right) the improvement or worsening of the dynamic (static) quality measure, for `Random`, `Central`, and `Leftmost Embedding Algorithm`, respectively, is shown.

`Leftmost-Embedding Algorithm`, which performs better than the other two in all experiments that we have run.

The charts relative to `Leftmost-Embedding Algorithm` illustrate tradeoffs between the values of the dynamic and static quality measures. In particular, as k increases, there is an improvement of the performance with respect to $m_D(h, k)$ and a worsening with respect to $m_S(h, k)$. We can summarize the outcome of our experimental study as follows. Each level of knowledge identifies a tradeoff between the aesthetics of the drawing and the preservation of the mental map. An effective approach to the design of a drawing algorithm that achieves a pleasing compromise between aesthetic requirements and preservation of the mental map is based on using only a limited knowledge of the future.

4 Analysis

In this section we analytically compare the performance of `Leftmost-Embedding Algorithm` and of `Central-Embedding Algorithm`. This comparison considers the behavior of the total-shift quality measures for both algorithms when k goes to infinity. The analytical behavior of the curves of $m_D(h,k,t)$ for large values of k is consistent with the behavior observed experimentally for small values of k. We start by introducing some definitions and basic tools. Let v be a vertex of $T_{h,k}(t)$ and consider a drawing Γ of $T_{h,k}(t)$. The difference between the x-coordinate of v and the x-coordinate of $p(v)$ in Γ is denoted as $\delta_{h,k}(v,t)$, i.e., $\delta_{h,k}(v,t) = x(p(v),t) - x(v,t)$. The following lemma shows the interplay between time and knowledge by relating the values of $\delta()$ at consecutive instants of time with the values of $\delta()$ for consecutive values of k.

Lemma 1. *Let v be a non-root vertex in $T_{h,k}(t) \cap T_{h,k}(t+1) = T_{h,k-1}(t+1)$. For any drawing produced by* `Leftmost-Embedding Algorithm` *(*`Central-Embedding Algorithm`*) we have $\delta_{h,k}(v,t) = \delta_{h,k-1}(v,t+1)$.*

Proof. $\delta_{h,k}(v,t)$ is measured on the drawing of $T_{h,k}(t)$, while $\delta_{h,k-1}(v,t+1)$ is measured on the drawing of $T_{h,k-1}(t+1)$. Because $T_{h,k-1}(t+1)$ is the subtree of $T_{h,k}(t)$ rooted at $r(t+1)$, and since `Leftmost-Embedding Algorithm` (`Central-Embedding Algorithm`) draws a subtree with a bottom-up-strategy independent of the size of the enclosing supertree, we have that $\delta_{h,k}(v,t) = \delta_{h,k-1}(v,t+1)$.

From the previous lemma we have that the variation of δ from instant t to instant $t+1$ is equal to the variation of δ when the knowledge at instant $t+1$ varies from $k-1$ to k.

Corollary 1. $\delta_{h,k}(v,t+1) - \delta_{h,k}(v,t) = \delta_{h,k}(v,t+1) - \delta_{h,k-1}(v,t+1)$.

Corollary 1 allows us to study the variation of δ by considering $T_{h,k-1}(t+1)$ and $T_{h,k}(t+1)$, instead of looking at $T_{h,k}(t)$ and $T_{h,k}(t+1)$. Since $T_{h,k-1}(t+1)$ and $T_{h,k}(t+1)$ share the root, we can univocally define a *level* for their vertices; we assume that the root has level 0.

4.1 Analysis of `Leftmost-Embedding Algorithm`

Consider a drawing Γ of $T_{h,k}(t+1)$ produced by `Leftmost-Embedding Algorithm`. All vertices that have the same level in $T_{h,k}(t+1)$ also have the same y-coordinate in Γ. Thus, we also talk about levels of Γ. Let $0 \le i \le k$ be a level of Γ, let $r(t+1+i)$ be the vertex of the visualization path at level i, and let u be its rightmost child. We denote with $\omega(i,k)$ the quantity $\omega(i,k) = x(u,t+1) - x(r(t+1+i),t+1)) = |\delta_{h,k}(u,t+1)|$. Also, we denote with $\mu(k)$ the quantity $\mu(k) = \omega(k,k)$. Let $1 \le i \le k$ be a level of Γ, let $r(t+i)$ be the vertex of the visualization path at level $i-1$, and let $r(t+1+i)$ its leftmost child. Also, let v be the rightmost child of $r(t+i)$ and let u be the rightmost child of $r(t+1+i)$. We denote with $\lambda(i,k)$ the quantity $\lambda(i,k) = x(v) - x(u)$. Examples of the quantities $\lambda(i,k)$, $\omega(i,k)$, and $\mu(k)$ are depicted in Figure 4.

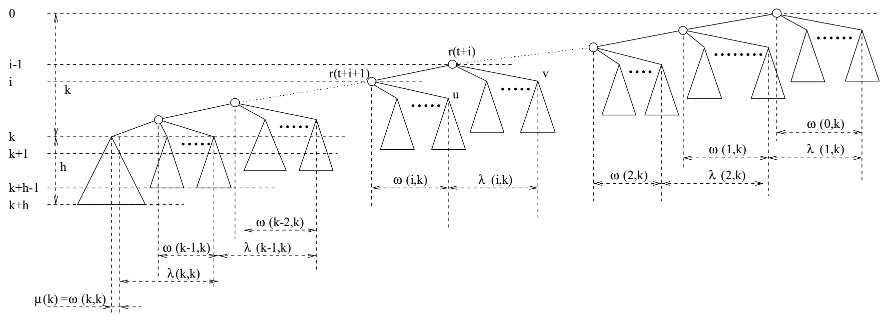


Fig. 4. Quantities $\lambda(i,k)$, $\omega(i,k)$, and $\mu(k)$ for the analysis of the drawings produced by `Leftmost-Embedding Algorithm`.

Lemma 2. *For any positive integer k and for any integer $0 \leq i \leq k-1$, we have that $\lambda(i,k) = \lambda(i,k-1)$.*

Lemma 3. *Let t be an instant of time, and let h and k be the integers that define the size of the visualization window and of the knowledge window. For a drawing of $T_{h,k}(t)$ produced by* `Leftmost-Embedding Algorithm` *we have that:*

$$\omega(i,k) = \frac{\mu(k)}{2^{k-i}} + \sum_{j=1}^{k-i} \frac{\lambda(i+j,k)}{2^j} \qquad\qquad i \in [0,k]$$

Proof. The proof follows from the solution of the following recurrence equation (see also Figure 4):

$$\omega(i,k) = \begin{cases} \frac{1}{2}[\omega(i+1,k) + \lambda(i+j,k)] & 0 \leq i < k \\ \\ \mu(k) & i = k \end{cases}$$

We are now ready to study the variation of ω as k goes to infinity. In order to ensure a finite limit, we need to bound the values of $\mu(k)$, $\mu(k-1)$, and $\lambda(k,k)$. Lemma 4 shows that this is always possible under the assumption that the maximum vertex degree of T is finite. The limit for the variation of ω as k goes to infinity is computed in Lemma 5.

Lemma 4. *Let T be an infinite tree whose maximum vertex outdegree is bounded by a constant. Then there exists a constant ξ such that $\forall k$ $\lambda(k,k) - \mu(k) + 2\mu(k-1) \leq \xi$.*

Lemma 5. *Let T be an infinite tree whose maximum vertex outdegree is bounded by a constant. Then,*

$$\forall i \in [0,k] \qquad \lim_{k\to\infty} (\omega(i,k) - \omega(i,k-1)) = 0$$

Proof. In view of Lemma 4, a constant ξ exists such that $\forall k\ \lambda(k,k) - \mu(k) + 2\mu(k-1) \leq \xi$. The claim immediately follows from Lemma 2 and Lemma 3:

$$\lim_{k\to\infty} (\omega(i,k) - \omega(i,k-1)) = \lim_{k\to\infty} \frac{\lambda(k,k) - \mu(k) + 2\mu(k-1)}{2^{k-i}} \leq \lim_{k\to\infty} \frac{\xi}{2^{k-i}} = 0$$

The following lemma provides a concise formula for the sum of the contributes to the dynamic quality measure $m_D(h,k,t)$ due to the vertices laying on the same level of $T_{h,k}(t)$.

Lemma 6. *Let t be an instant of time, and let h and k be the integers that define the size of the visualization window and of the knowledge window. For a drawing of $T_{h,k}(t)$ produced by* `Leftmost-Embedding Algorithm`*, we have that $\forall i \in [0, h-2]$:*

$$\sum_{\substack{v \in T_{h-1}(t+1) \\ \text{at level } i+1}} |\delta_{h,k-1}(v,t+1) - \delta_{h,k}(v,t+1)| = outdeg(r_i)(\omega(i,k) - \omega(i,k-1))$$

We are now able to express the dynamic quality measure as a function of $\omega(i,k)$ and to compute the limit of the improvement $i_D(h,k,t) = \frac{m_D(h,0,t) - m_D(h,k,t)}{m_D(h,0,t)}$ as k goes to infinity.

Theorem 1. *Let t be an instant of time, and let h and k be two integers that define the size of the visualization window and of the knowledge window. For a drawing of $T_{h,k}(t)$ produced by* `Leftmost-Embedding Algorithm`*, let $m_D(h,k,t)$ be the total-shift dynamic quality measure. Then*

$$\lim_{k\to\infty} \frac{m_D(h,0,t) - m_D(h,k,t)}{m_D(h,0,t)} = 1$$

Proof. We recall from Section 2 the definition of the dynamic quality measure:

$$m_D(h,k,t) = \sum_{v \in T_{h-1}(t+1) - \{r(t+1)\}} |x(p(v),t) - x(v,t) - (x(p(v),t+1) - x(v,t+1))|$$

According to the definition of δ we have that $x(p(v),t) - x(v,t) = \delta_{h,k}(v,t)$ and $x(p(v),t+1) - x(v,t+1) = \delta_{h,k}(v,t+1)$ and that, in view of Lemma 1, $\delta_{h,k}(v,t) = \delta_{h,k-1}(v,t+1)$. Hence

$$m_D(h,k,t) = \sum_{v \in T_{h-1}(t+1) - \{r(t+1)\}} |\delta_{h,k-1}(v,t+1) - \delta_{h,k}(v,t+1)|$$

Partitioning vertices in $T_{h-1}(t+1)$ according to their level we have

$$m_D(h,k,t) = \sum_{i=0}^{h-2} \sum_{\substack{v \in T_{h-1}(t+1) \\ \text{at level } i+1}} |\delta_{h,k-1}(v,t+1) - \delta_{h,k}(v,t+1)|$$

Then, in view of Lemma 6, and supposing $k \geq h$ in order to use ω in its domain:

$$m_D(h,k,t) = \sum_{i=0}^{h-2} outdeg(r(t+1+i))|\omega(i,k) - \omega(i,k-1)|$$

Therefore, the truth of the statement follows from Lemma 5.

4.2 Analysis of `Central-Embedding Algorithm`

In this section we compare the behavior of `Central-Embedding Algorithm` to that of `Leftmost-Embedding Algorithm` with respect to the total-shift dynamic quality measure. Consider a drawing of $T_{h,k}(t+1)$ produced by `Central-Embedding Algorithm`. The quantities $\omega(i,k)$ and $\mu(k)$ can be defined in the same way as in Subsection 4.1. We need two new definitions. Refer also to Figure 5. Let $r(t+1+i)$ be a vertex at level $1 \leq i \leq k$ in the visualization path and let $p(r(t+1+i))$ be its parent. Let u and v be the rightmost and the leftmost children of $r(t+1+i)$, respectively. Let w and z be the rightmost and the leftmost children of $p(r(t+1+i))$, respectively. We denote with $\lambda^l(i,k)$ the quantity $\lambda^l(i,k) = x(v) - x(z)$ and with $\lambda^r(i,k)$ the quantity $\lambda^r(i,k) = x(w) - x(u)$.

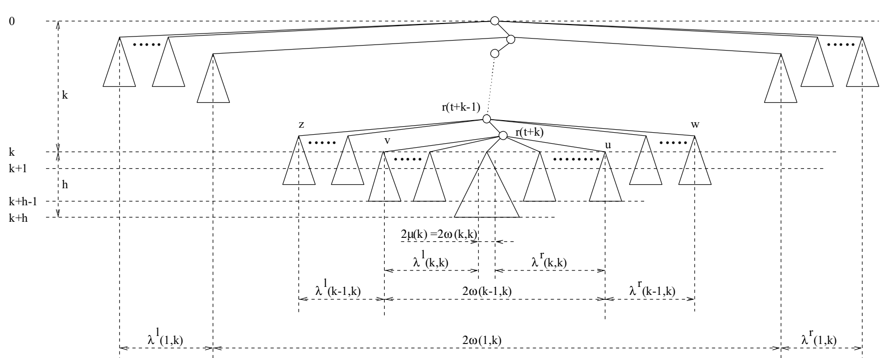


Fig. 5. Pruned tree and quantities $\lambda^l(i,k)$, $\lambda^r(i,k)$, $\omega(i,k)$, and $\mu(k)$ for the analysis of the drawings produced by `Central-Embedding Algorithm`.

With reasoning similar to that of Lemma 3 and Lemma 6, the following lemmas can be proved.

Lemma 7. *Let t be an instant of time, and let h and k be the integers that define the size of the visualization window and of the knowledge window. For a drawing of $T_{h,k}(t)$ produced by* `Central-Embedding Algorithm` *we have that:*

$$\omega(i,k) = \mu(k) + \sum_{j=1}^{k-i} \frac{\lambda^l(i+j,k) + \lambda^l(i+j,k)}{2} \qquad i \in [0,k]$$

Lemma 8. *Let t be an instant of time, and let h and k be the integers that define the size of the visualization window and of the knowledge window. For a drawing of $T_{h,k}(t)$ produced by* `Central-Embedding Algorithm` *we have that $\forall i \in [0, h-2]$:*

$$\sum_{\substack{v \in T_{h-1}(t+1) \\ at\ level\ i+1}} |\delta_{h,k-1}(v, t+1) - \delta_{h,k}(v, t+1)| = [outdeg(r_i) - 1](\omega(i,k) - \omega(i, k-1))$$

By means of Lemma 6 and Lemma 8, we can prove the following theorem, which states that for any sufficiently large value of k the performances of `Leftmost-Embedding Algorithm` are always better than the performances of `Central-Embedding Algorithm` .

Theorem 2. *Let t be an instant of time, and let h and k be the integers that define the size of the visualization window and the size of the knowledge window. Let ${m_D}^l(h,k,t)$ and ${m_D}^c(h,k,t)$ be the total-shift dynamic quality measure of* `Leftmost-Embedding Algorithm` *and* `Central-Embedding Algorithm`*, respectively. Then $\exists k_0 \geq 0 : \forall k \geq k_0, {m_D}^c(h,k,t) \geq {m_D}^l(h,k,t)$*

References

1. *Astra*. Mercury international. `http://www.merc-int.com/`.
2. U. Brandes and D. Wagner. A bayesian paradigm for dynamic graph layout. In G. Di Battista, editor, *Graph Drawing (Proc. GD '97)*, volume 1353 of *Lecture Notes Comput. Sci.*, pages 236–247. Springer-Verlag, 1998.
3. R. F. Cohen, G. Di Battista, R. Tamassia, and I. G. Tollis. Dynamic graph drawings: Trees, series-parallel digraphs, and planar *ST*-digraphs. *SIAM J. Comput.*, 24(5):970–1001, 1995.
4. P. Crescenzi, C. Demetrescu, I. Finocchi, and R. Petreschi. Leonardo: a software visualization system. In G. Italiano and S. Orlando, editors, *Workshop on Algorithm Engineering*, pages 146–155, 1997.
5. G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. *Graph Drawing.* Prentice Hall, Upper Saddle River, NJ, 1999.
6. C. A. Duncan, M. Goodrich, and S. Kobourov. Balanced aspect ratio trees and their use for drawing very large graphs. In S. Whitesides, editor, *Graph Drawing (Proc. GD '98)*, volume 1547 of *Lecture Notes Comput. Sci.*, pages 384–393. Springer-Verlag, 1999.
7. P. Eades, R. F. Cohen, and M. L. Huang. Online animated graph drawing for web navigation. In G. Di Battista, editor, *Graph Drawing (Proc. GD '97)*, volume 1353 of *Lecture Notes Comput. Sci.*, pages 330–335. Springer-Verlag, 1997.
8. P. Eades, W. Lai, K. Misue, and K. Sugiyama. Preserving the mental map of a diagram. In *Proceedings of Compugraphics 91*, pages 24–33, 1991.
9. T. R. Henry and S. E. Hudson. Viewing large graphs. Technical Report 90-13, Department of Computer Science, University of Arizona, 1990.
10. *Hyperbolic Tree.* Inxight. `http://www.hyperbolictree.com/`.
11. K. Kaugars, J. Reinfelds, and A. Brazma. A simple algorithm for drawing large graphs on small screens. In R. Tamassia and I. G. Tollis, editors, *Graph Drawing (Proc. GD '94)*, volume 894 of *Lecture Notes Comput. Sci.*, pages 278–281. Springer-Verlag, 1995.

12. D. Kimelman, B. Leban, T. Roth, and D. Zernik. Reduction of visual complexity in dynamic graphs. In R. Tamassia and I. G. Tollis, editors, *Graph Drawing (Proc. GD '94)*, volume 894 of *Lecture Notes Comput. Sci.*, pages 218–225. Springer-Verlag, 1995.
13. E. B. Messinger, L. A. Rowe, and R. H. Henry. A divide-and-conquer algorithm for the automatic layout of large directed graphs. *IEEE Trans. Syst. Man Cybern.*, SMC-21(1):1–12, 1991.
14. K. Misue, P. Eades, W. Lai, and K. Sugiyama. Layout adjustment and the mental map. *J. Visual Lang. Comput.*, 6(2):183–210, 1995.
15. S. Moen. Drawing dynamic trees. *IEEE Softw.*, 7:21–28, 1990.
16. T. Munzner. Exploring large graphs in 3d hyperbolic space. *Comp. Graphics and its Applications*, 8(4):18–23, 1998.
17. T. Munzner. Drawing large graphs with h3viewer and site manager. In S. Whitesides, editor, *Graph Drawing (Proc. GD '98)*, volume 1547 of *Lecture Notes Comput. Sci.*, pages 384–393. Springer-Verlag, 1999.
18. S. North. Incremental layout in DynaDAG. In F. J. Brandenburg, editor, *Graph Drawing (Proc. GD '95)*, volume 1027 of *Lecture Notes Comput. Sci.*, pages 409–418. Springer-Verlag, 1996.
19. E. Reingold and J. Tilford. Tidier drawing of trees. *IEEE Trans. Softw. Eng.*, SE-7(2):223–228, 1981.
20. G. J. Wills. NicheWorks - interactive visualization of very large graphs. In G. Di Battista, editor, *Graph Drawing (Proc. GD '97)*, volume 1353 of *Lecture Notes Comput. Sci.*, pages 403–414. Springer-Verlag, 1997.

Latour — A Tree Visualisation System

Ivan Herman[1], Guy Melançon[1], Maurice M. de Ruiter[1], and Maylis Delest[2]

[1] Centrum voor Wiskunde en Informatica, P.O. Box 94079
1090 GB Amsterdam, The Netherlands
{Ivan.Herman,Guy.Melancon, Behr.de.Ruiter}@cwi.nl
[2] LaBRI, Université Bordeaux I
351, cours de la Libération, 33405 Talence Cedex, France
Maylis.Delest@labri.u-bordeaux.fr

Abstract. This paper presents some of the most important features of a tree visualisation system called Latour, developed for the purposes of information visualisation. This system includes a number of interesting and unique characteristics, for example the provision for visual cues based on complexity metrics on graphs, which represent general principles that, in our view, graph based information visualisation systems should generally offer.

1 Introduction

Information visualisation is one of the relatively new areas of research and development in computer science; its fundamental goal, i.e., the ability to visualise and to navigate in large, abstract datastructures, is often regarded as one of the crucial tasks in bringing computers closer to the general public [2]. Visualising graphs plays a very special role in this area, because they can often be used to visualise abstract datastructures. Practical examples include hypermedia structures (like the Web), database query results, or organisational charts of companies. Systems to visualise large graphs have come to the fore in the last years; the NicheWorks system of Wills [17], the fsviz system of Carrière and Kazman [3], or daVinci of the University of Bremen [6] are just some typical examples. These systems usually draw on the rich research heritage in the graph drawing community which, over the years, has explored some of the mathematical problems related to graph drawings [1]. Putting these research results into practice is not a simple task, however. Practical issues raised by, for example, the large size of graphs in information visualisation, the need for navigation and interaction, user interface and ergonomic issues, etc., create new challenges, or cast a new light on well–accepted practices [13]. Consequently, none of the current graph drawing systems could claim to be complete; experiences with these systems are still to be gathered to gain a better understanding of the kind of drawing and navigation facilities which are necessary for a really successful system.

The goal of this paper is to contribute to this "gathering". It describes an application framework called Latour, whose goal is to incorporate interactive

J. Kratochvíl (Ed.): GD'99, LNCS 1731, pp. 392–399, 1999.

graph (primarily tree) visualisation and navigation techniques into other applications. At present, Latour is used or tested as a toolkit to visualise, and to interact with, various application data, for example internal data structures of programs, deployment results of large Petri nets, evolution of genetic algorithms, etc.

While developing this framework, some of the practical problems required more concentrated research efforts, which also led to interesting and general results [8,9,12]. The goal of this paper is to describe a number of issues which, albeit not deserving separate articles by themselves, together constitute a body of experiences which we felt is worth sharing with the R&D community. A technical report available online [10] contains further details which, because of lack of place, could not be included in the present paper.

2 Graph/Tree Layout

In spite of all the results on graph drawing [1], it is not simple to choose a specific algorithm for information visualisation. Information visualisation, which is inherently interactive, raises a number of issues that are not necessarily covered by the classical research. Apart from obvious problems such as speed (in the case of a graph with 3–4000 nodes, the display of the graph should not take more than a second), there remain two important aspects:

- Predictability. Two different runs of the algorithm, involving the same or similar graphs, should not lead to radically different visual representation. This is very important if the graph is interactively changed, for example by (temporarily) hiding some nodes or making them visible again. Great care should be taken on which layout algorithm is chosen. For example, a number of graph layout algorithms use optimisation techniques; if the graph changes, a new local minimum may lead to a dramatically different visual representation, which is unacceptable for interactive use. (The term "preserving the mental model" is also used to describe this requirement, see [11].)
- Navigation on large or unusual graphs. Practical applications lead to thousands, or possibly tens of thousands of nodes. To cope with such numbers, navigation tools, search facilities, hierarchical views, etc., are necessary. The implementation of such tools may also require the usage of suboptimal layout algorithms.

The bulk of the Latour system concentrates on trees, where the usual layout algorithms are quite predictable and fast. It was not the goal of Latour to develop new layout algorithms; instead, the goal was to concentrate on the issues raised by data exploration and interaction. Three different tree layout algorithms have been implemented. Various user communities have their own traditions, habits, or requirements, and an application framework cannot impose one single view on its users. In what follows, a short overview of these views will be given.

2.1 Hierarchical View

The hierarchical view of the tree is based on the well–known algorithm of Reingold and Tilford [14] revisited by Walker [16]. The layout algorithm is simple, fast, and completely predictable. Various variants exist: grid–based, left–to–right or top–down, etc. All these variations are mathematically identical and implementers may be tempted to include arbitrarily one of these variations only. This would be a mistake: one should recognise that the way of looking at trees may depend on the application areas. For example, the top–down grid view is the widespread way of looking at family trees, whereas biological evolution schemas often use a left–to–right grid. The conclusion is simple, albeit important: give the user the choice; he/she should be able to choose among the different views.

2.2 Radial View

The radial view is based on an algorithm described in Eades [5] (see also [1]). This algorithm recursively places the children of a subtree into circular wedges; the central angle of these wedges are proportional to the width of the respective subtrees, i.e., the number of leaves. If this was the only layout rule, additional edge intersections would occur if the angle on the node became too large; to avoid this, a “convexity constraint” is introduced which, essentially, forces the wedge to remain convex. This type of view is favoured, for example, by some web site viewers, which do not want to overemphasise the role of a root.

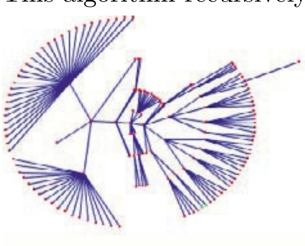


Fig. 1: Radial view without convexity check

The algorithm is very simple, but it is not optimal in using the available space. We spent some time in trying to optimise the algorithm. The idea was to use the statistical distribution of the width of a subtree at a node, which can be approximated with a normal distribution (see [4]). The improvements were not significant, however; this turned out to be the consequence of the convexity constraint whose effect seems to dominate other optimisation attempts. A possibility to overcome this problem is to simply drop the convexity check. Although this is not mathematically correct, the occurrence of extra intersections is not very frequent after all. It is not necessary to look for a mathematically perfect algorithm for a graph layout; the mathematical “faults” may not be significant in practice. Problems with navigation, zooming, etc. (see the next section) should become predominant in that case, and it is not really worth to optimise the layout any further. For the sake of completeness, we decided to include both the optimal (i.e., with convexity check) and the, shall we say, sub–optimal radial layout algorithm into Latour. See [10] for a comparison of both layouts including figures.

2.3 Balloon View

The request for a "balloon" view (see Fig. 2) came from an application dealing with the retrieval of keywords and their relations from a database. The notion of a "root" is temporary for such application: the user should be able to move from one node to the other interactively, and the tree on the screen should reflect the relationships using this temporary focus. The balloon view seems to fulfil this need. The detailed explanation of the algorithm would go beyond the scope of this paper [12]. Other placement algorithms could also be used [3].

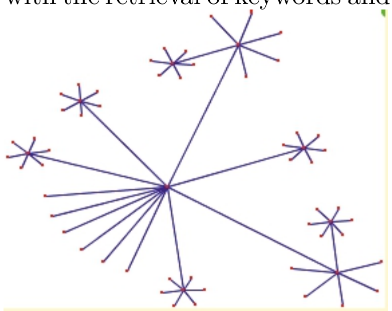


Fig. 2: Balloon view

3 Interaction and Navigation

Information visualisation is an inherently interactive application; the user has to move around in information space, explore details, hide unnecessary parts of a tree, etc. Obviously, a good system must offer a whole range of tools in order to make the exploration of a graph easy, or indeed possible.

3.1 Zoom, Pan, Fish–Eye

Some of the techniques, implemented in Latour are now standard: zoom, pan, and geometric fish–eye [15]. As much as possible, the factors controlling these effects (e.g., the distortion factor of the fish–eye view) are settable interactively by the end–user.

The fish–eye view has one drawback, though, which implementers should be aware of. The essence of a fish–eye view is to distort the position of each node, using a concave function applied on the distance between the focal point and the node's position. If the distortion were to be applied faithfully, the edges connecting the nodes should be distorted into general curves. Usual graphics systems do not offer the necessary facilities to draw these curves easily. The implementer's only choice is to approximate these curves with dense polylines. This leads to a prohibitively large amount of calculation and makes the responsiveness of the system sink to an unacceptably low level. The only viable solution is to apply the fish–eye distortion on the nodes only, and to connect them by straight–line edges. The consequence of this inexact solution is that new edge intersections might occur. Though inelegant, this brute force approach did not prove to be disturbing in practice.

3.2 Complexity Visual Cues

The well–known problem in using zoom and pan is that the user looses the "context". This is why fish–eye view is used: it provides a "focus+context" view

of the tree. However, when the tree is large, zoom and pan cannot be avoided and other techniques become necessary, too. A unique feature of Latour is a technique to provide visual cues based on the structural complexity of the tree. This technique works as follows.

A metric value is calculated for each node of the tree. This metric should represent the complexity of the subtree stemming at the node. Several different metric functions are possible and, ultimately, the choice among these should be application dependent. Examples for such metric include the width of the subtree (i.e., the number of leaves), the sum of the lengths of all paths between the node and the leaves of the subtree, the so–called Strahler numbers [8], or the "degree of interest" function used by Furnas [7]. Using these metric values, and visual tools like colour saturation, linewidth, etc, Latour can highlight the "backbone" of a tree, i.e., those edges which hold larger, more complex subtrees. The effect is clearly visible on Fig. 3. Without the backbone the user would barely know where to move with the pan, if complex areas are searched. The backbone on the figure clearly shows, for example, that one of the edges going toward the left leads to a complex portion of the tree, whereas the other ones are probably less interesting.

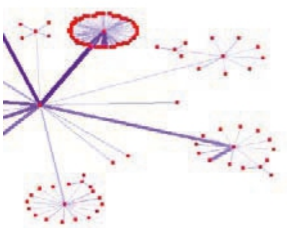


Fig. 3: Visual Cues

Another possible usage of the metric numbers is presented on Fig. 4: this is the so–called schematic view of a tree. Based on the complexity metrics of the nodes, Latour displays only those nodes whose metric value is greater than a specific cut–off, yielding what we have called the skeleton of the tree. All other nodes are encapsulated in triangular shapes, whose size and geometry is proportional to the hidden portion of the tree. The result is a better overall view of the tree which, combined with other navigation techniques, provides a powerful interactive tool. It is worth noting that, although all our examples so far were for trees, the visual cue techniques based on a complexity metric represent a general principle which can be applied for more general graphs, too; the interested reader should refer to [9].

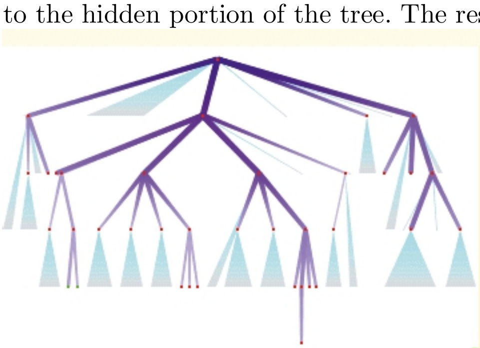


Fig. 4: Schematic view of a tree

3.3 Animation

Latour is an interactive system; the user navigates in different portions of the tree, zooms, pans, etc. Some of these actions result in an immediate, real–time feedback (for example, zoom), some other actions may lead to a more radical reorganisation of the screen (for example, folding a subtree into a node, or unfolding a folded subtree). Latour animates all possible changes from one view to the

other, avoiding any radical changes as far as possible. Although originally only included in Latour to reduce possible ergonomic problems, this basic animation feature turned out to be a very useful tool for various applications exploring a sequence of trees, instead of a single one. This is the case, for example, of the application exploring genetic algorithms, or the traces of parallel program runs. Therefore, the input possibilities of Latour have been extended: it can not only accept the description of a single tree, but also a "generation" of trees, i.e., a basic tree plus a sequence of difference trees. This sequence of trees can then be visualised systematically with again a graceful animation at each change.

4 Beyond Trees

Latour is primarily a tree visualisation tool but, obviously, applications may want to handle more general structures, too. We have added some extensions to Latour which are worth mentioning here.

4.1 Packed Forests

Packed forests are special data structures. The need for these data structures has arisen through an application concerned with the visualisation of the internal data structures of compilers, but has proven to be useful in general, too.

Instead of giving an abstract definition, the concept is presented through an example. For a compiler, the standard internal representation of a string is a list. The leaves of the list represent the individual characters of the string, and intermediate nodes are used to build up a list structure. Such list can be represented as a simple tree, like the left–hand one in Fig. 5. However, such a representation may be too "verbose". An expert in compiler technology knows the internal representation for a string and does not necessarily need the full list version of the relevant portion of the graph; the tree on the right–hand side of Fig. 5 is enough to convey all the necessary information. What the user wants is to be able to interactively "switch" between different representations. Latour has the possibility to store, internally, a set of such alternatives for each node, and offers interactive means to switch among those.

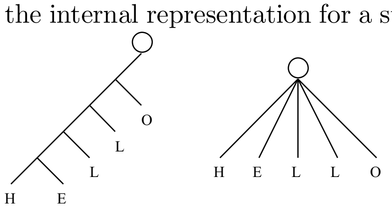


Fig. 5: A packed forest

Packed forests turned out to be extremely useful in practice. As a slightly extreme example, some of the demonstration graphs used by our compiler builder partner is, initially, a tree consisting of 2–3 nodes only. However, when the same graph has all its most complex alternatives extended, it turns into a tree of about 100 nodes. Similar data structures are used routinely in computational linguistics; the concept of "level of details", of an utmost importance in virtual reality scenes, is another example which can be represented through these

structures. Packed forests provide a very efficient way of imposing a manageable hierarchy on the visualised data structures.

4.2 Dag's

Dag's (Directed Acyclic Graphs) represent the next logical step when trying to generalise from trees. This is achieved by a simple extension of Latour, which allows the storage of additional links for each node of the underlying tree. This means that a spanning tree is provided, and Latour uses its tree–related structure to visualise the dag by simply adding the additional links to the tree picture. The spanning tree may have two origins: either the application generates it, or the spanning is tree is calculated for the dag.

Requesting the application to generate a spanning tree is not such a strong requirement. A number of applications have an inherent tree structure in the data, and visualising this tree, with the additional edges added to the tree, yields a natural representation of the dag.

Fig. 6 shows an example where a spanning tree is used to visualise a dag. The interesting feature in this case is that the spanning tree consists of three branches and most of the "non–tree" edges are used to connect nodes in different branches. We can refer to such graphs as "multipartite" trees. Similar, but bipartite trees occur when describing virtual reality scenes, for example (where one branch describe an object hierarchy, the other the real instances). These "multipartite" graphs occur frequently in applications, and constitute a set of examples where the simple extension of Latour works out very well in practice. Automatic generation of spanning trees raises a number of issues not developed here. For further details see [10].

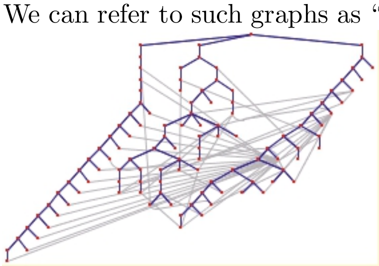


Fig. 6: A tree with added links

5 Conclusions

The implementation of Latour has resulted in a very flexible system, which is well adaptable to various user communities. It concentrates on interaction and visual feed–back, rather than complicated layout algorithms, which makes it one of its strengths. It has also taught us some important lessons: that a proper balance has to be found between the mathematical correctness and the requirements of navigation and interaction, that the end–user has to have a maximal control over the appearance and the attributes of the visual representation, we learned about the importance of metric functions on graphs in general. In developing a more general graph–based information visualisation framework these experiences will become of an utmost importance.

References

1. Battista, G. di, Eades, P., Tamassia, R., Tollis, I.G.: *Graph drawing: algorithms for the visualisation of graphs.* Prentice Hall (1999).
2. Card, S.K., Mackinlay, J.D., Shneiderman, B. (eds): *Readings in Information Visualization.* Morgan Kaufmann Publishers (1999).
3. J. Carrière, J., Kazman, R.: Interacting with huge trees: beyond cone trees. In: *Proceedings IEEE Information Visualisation '95*, IEEE CS Press (1995), 74–81.
4. Drmota, M.: Systems of functional equations. In: *J. Random Structures and Algorithms*, **10**(1–2), (1997), 103–124.
5. Eades, P.: Drawing free trees. In: *Bulletin of the Institute for Combinatorics and its Applications*, **5**, (1992), 10–36.
6. Fröhlich, M., Werner, M.: Demonstration of the interactive graph visualization system daVinci. In: *Proceedings of DIMACS Workshop on Graph Drawing '94, Springer–Verlag*, (1995).
7. Furnas, G.W.: Generalized fisheye views. In: *Human Factors in Computing Systems, CHI'95 Conference Proceedings*, ACM Press (1995), 16–23.
8. Herman, I., Delest, M., Melançon, G.: Tree visualisation and navigation clues for information visualisation. In: *Computer Graphics Forum*, **17**(2), (1998), 153–165.
9. Herman, I., Marshall, S.M., Melançon, G., Duke, D.J., Delest, M., Domenger, J.–P.: Skeletal images as visual cues for graph visualisation. In: *Data Visualization '99, Proceedings of the Joint Eurographics IEEE TCVG Symposium on Visualization*, Springer–Verlag, (1999), 13–22.
10. Herman, I., Melançon, G., Ruiter, M.M. de, Delest, M.: *Latour — a tree visualisation system.* Reports of the Centre for Mathematics and Computer Sciences (CWI), INS–R9904, `http://www.cwi.nl/InfoVisu/papers/LatourOverview.pdf`, (1999).
11. Misue, K.,Eades, P., Lai W., Sugiyama, K.: Layout adjustment and the mental map. In: *Journal of Visual Languages and Computing*, **6**, (1995), 183–210.
12. Melançon, G., Herman, I.: *Circular drawing of rooted trees.* Reports of the Centre for Mathematics and Computer Sciences (CWI), INS–R9817, `http://www.cwi.nl/InfoVisu/papers/circular.pdf`, (1998).
13. Purchase, H.: Which Aesthetic has the Greatest Effect on Human Understanding? In: *Proceedings of the Symposium on Graph Drawing GD'97*, Springer–Verlag (1998), 248–261.
14. Reingold, E.M., Tilford, J.S.: Tidier drawing of trees. In: *IEEE Transactions on Software Engineering*, **SE–7**(2), (1981), 223–228.
15. Sarkar, M., Brown, M.H.: Graphical fisheye views. In: *Communication of the ACM*, **37**(12), (1994), 73–84.
16. Walker II, J.Q.: A node–positioning algorithm for general trees. In: *Software — Practice and Experience*, 20(7), (1990), 685–705.
17. Wills, G.J.: Niche Works — interactive visualization of very large graphs. In: *Proceedings of the Symposium on Graph Drawing GD'97*, Springer–Verlag (1998), 403–415.

Graph-Drawing Contest Report

Franz J. Brandenburg[1], Michael Jünger[2], Joe Marks[3],
Petra Mutzel[4], and Falk Schreiber[1]

[1] Universität Passau, 94030 Passau, Germany
brandenb,schreibe@fmi.uni-passau.de
[2] Universität zu Köln, Pohligstr. 1, D-50969 Köln, Germany
mjuenger@informatik.uni-koeln.de
[3] MERL–A Mitsubishi Electric Research Laboratory
201 Broadway Cambridge, MA 02139, U.S.A.
marks@merl.com
[4] Max-Planck-Institut für Informatik
Im Stadtwald, D-66123 Saarbrücken, Germany
mutzel@mpi-sb.mpg.de

Abstract. This report describes the Sixth Annual Graph Drawing Contest, held in conjunction with the 1999 Graph Drawing Symposium in Prague, Czech Republic. The purpose of the contest is to monitor and challenge the current state of the art in graph-drawing technology [2,3,5,6,4].

1 Introduction

Text descriptions of the four categories for the 1999 contest are available via the World Wide Web (WWW) [7]. Eighteen separate submissions were received (one more than last year), containing 57 different graph drawings (a record number), and one live demonstration. The winners for Categories A–C were selected by the contest organizers. The winners for Category D were ranked by the vote of all the symposium attendees. Conflicts of interest were avoided on an honor basis. The winning entries are described below.

2 Winning Submissions

2.1 Category A

The graph for Category A was unusually complex for the contest. It represents the characters in the long-running German soap opera, *Lindenstrasse*. This show started 14 years ago on ARD (one of the main public TV stations); every week about seven million people watch it. The data in the graph have been extracted from a poster in a popular book [8]. The following data are associated with each character node in the graph: the character's name and gender, the numbered episodes in which the character appears, the dates of the character's birth and death (if known), and the character's current status in the show (active, inactive, dead,

J. Kratochvíl (Ed.): GD'99, LNCS 1731, pp. 400–409, 1999.

or unknown). In addition, the importance of the character is also indicated, and the most important characters have pictures associated with them. The edges that relate characters are also typed to indicate the nature of the relationship: business, friendship, partnership, family, or hostility.

Three visualization tasks for this graph were mandated:

1. To depict the current situation.
2. To visualize the whole graph, including both current and past characters and relations.
3. To show the development of the graph over some time interval.

Only one submission was received for Tasks 1 and 3; none was received for Task 2. Nevertheless, the quality of the single submission was such that the committee decided to award the first-place prize to its creators, Vladimir Batagelj and Andrej Mrvar ([Vladimir.Batagelj, Andrej.Mrvar]@uni-lj.si) from the University of Ljubljana, Slovenia. The drawing in Figure 1 illustrates the current situation in the show.[1] It was drawn using the "Pajek" system [10]. An initial layout was computed using the Kamada-Kawai force-directed algorithm; this layout was subsequently edited by hand to arrange the nodes on a rectangular net. The dynamic drawings (not shown) were also computed using an incremental force-directed method.

2.2 Category B

Category B was this year's "theory surprise." It consisted of a directed graph whose edges were colored blue or green. No further information was provided. The 10 submissions received included an astonishing 49 different drawings of the graph!

The graph is the six-bit shift-register graph. It also occurs as the state graph of the minimal deterministic finite state automaton that recognizes the regular set $\{0|1\}^*1\{0|1\}^6$. Thus, the nodes are labeled most meaningfully as six-bit numbers from 000000 to 111111. The blue edges describe the left shifts from nodes with labels 0x. They define a binary tree with a self-loop at the root 000000. Similarly, the green edges describe the left shifts from nodes with labels 1x; they define a binary tree with a self-loop at the root 111111. These two trees are glued together. Conversely, the shift register graphs can be partitioned into two edge-disjoint trees. However, it is a bad strategy to draw shift-register graphs using a common tree-drawing algorithm for one of these trees, and then to insert the remaining edges. The shift-register graphs are highly symmetric and some symmetry should be displayed in any reasonable drawing.

Symmetry is indeed visible in the joint third-prize winners, shown in Figures 2–4. The drawing in Figure 2 was submitted by Karlis Freivalds and Paulis Kikusts (karlisf,paulis@cclu.lv), from the University of Latvia, Latvia. It was

[1] The original version of this drawing is in color, as are all of the winning drawings. A copy of this report with color drawings can be obtained from Joe Marks, marks@merl.com.

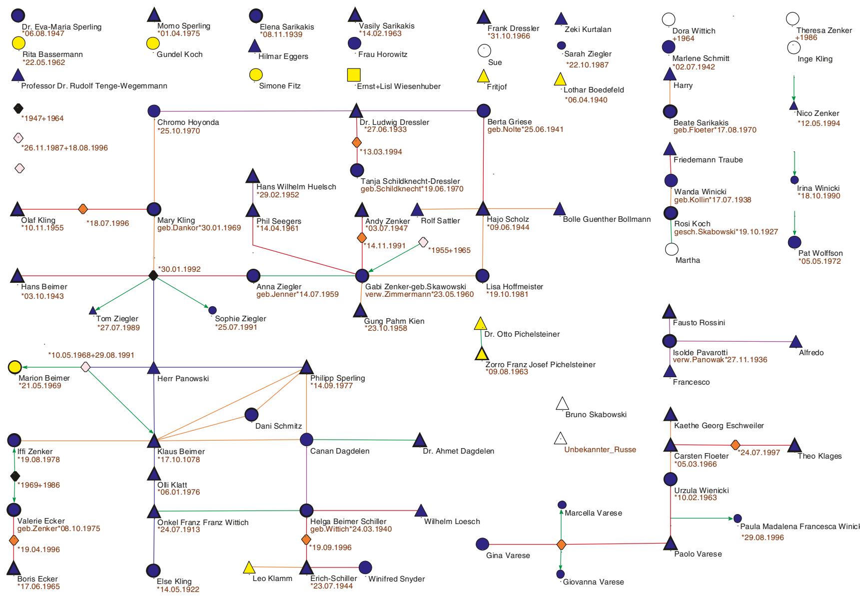


Fig. 1. First place, Category A (original in color).

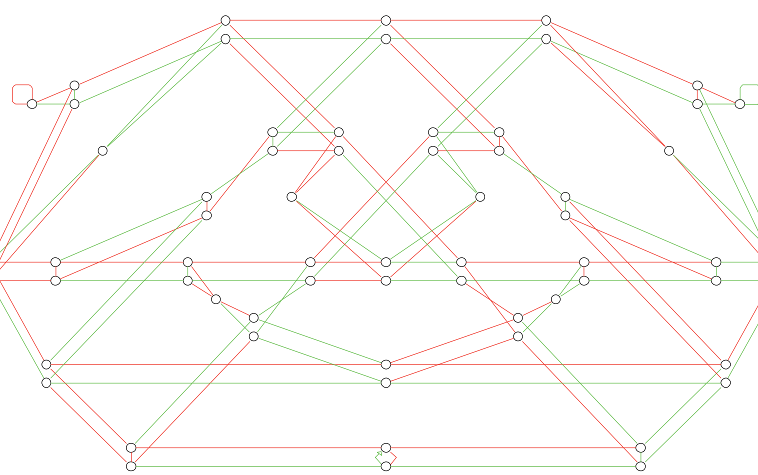


Fig. 2. Joint third place, Category B (original in color).

generated using the ActiveX Graphical Diagramming Engine (`http://www.-gradetools.com/`). The structure of the graph was discovered using a variant of barycentric layout, and subsequent refinement of node and edge positions was done by hand. Note the resemblance to a "smiley face"!

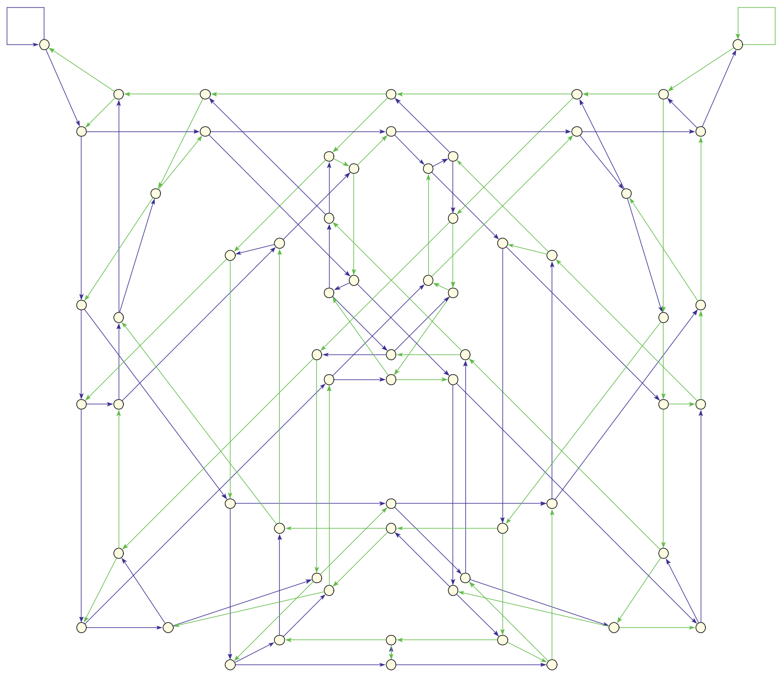


Fig. 3. Joint third place, Category B (original in color).

Ago Kuusik (Ago.Kuusik@ul.ie), from the University of Limerick, Ireland, submitted the drawing in Figure 3. An initial layout was computed using the 3D spring embedder of LEDA GraphWin. The layout was then refined manually.

The drawing in Figure 4 is called "Fool's Crown" by its author, Roland Wiese (wiese@informatik.uni-tuebingen.de), from the Universität Tübingen. The drawing was generated in the following steps:

- First the two nodes with self-loops were moved manually to opposite sides and fixed there.
- Then the GEM spring embedder was applied to the graph.
- Finally, the drawing was beautified via manual adjustment.

The third-place winners all required some manual editing to produce their final drawings. The second-place drawing for Category B required no manual adjustment, and is shown in Figure 5. It was produced by Vladimir Batagelj and

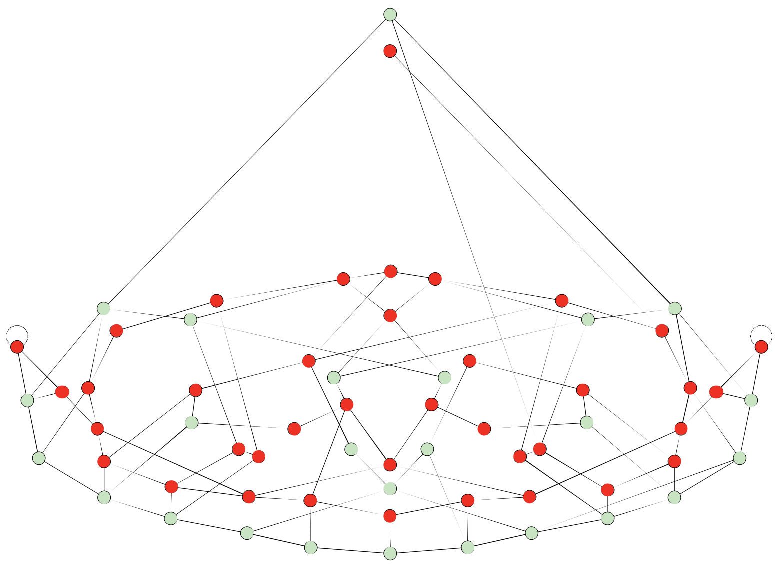


Fig. 4. Joint third place, Category B (original in color).

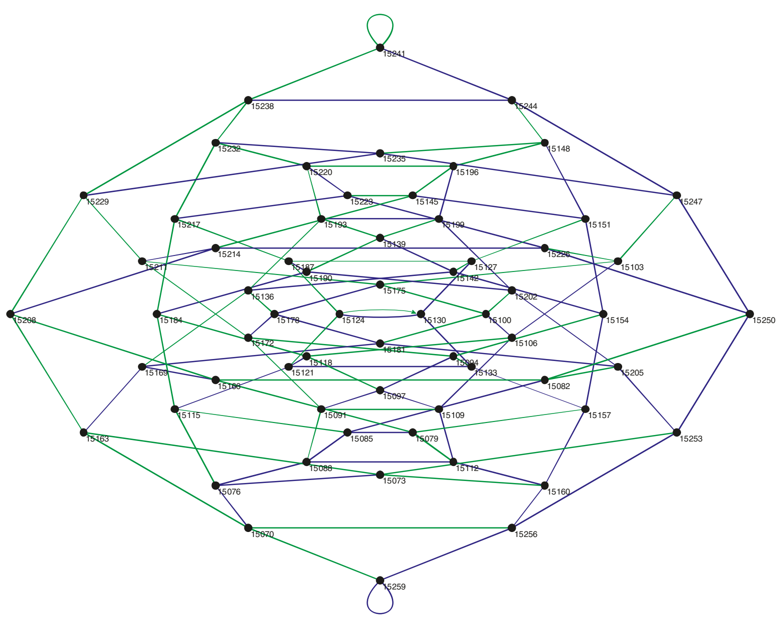


Fig. 5. Second prize, Category B (original in color).

Andrej Mrvar ([Vladimir.Batagelj, Andrej.Mrvar]@uni-lj.si) from the University of Ljubljana, Slovenia, using the “Pajek” system [10]. The layout was computed using a kind of Principal Components’ Analysis: the first, second, and fifth eigenvectors of the Laplacian matrix were used to generate a 3D embedding, and then a suitable projection was selected to generate the final 2D drawing.

The winning entry for Category B was submitted by Ulrik Brandes (ub@cs.brown.edu), from Brown University. He was the only contestant to determine the true underlying structure of the graph and its origin. In addition to awarding first place to Brandes’s submission, the judges felt that the written account of his investigation was of such high quality as to warrant publication in full. It can be found elsewhere in this volume, and contains several drawings of the contest graph [1].

2.3 Category C

Biochemistry is becoming an important application area for graph drawing. Biochemical pathways represent the complex interconnections between reactants, products, and enzymes in cells. In graph-theoretic terms they are hypergraphs. The contest hypergraph comes from the “Biochemical Pathways” atlas [9], and represents the main biochemical reactions of cells. For the contest it is modeled as a regular graph with additional unlabeled white nodes representing hyperedges of high degree. The nodes in the graph are labeled, and colored according to type, e.g., green for carbohydrates, blue for amino acids, etc. The graph edges are also typed: edges can be undirected, unidirectional, or bidirectional.

The drawing in Figure 6 was awarded joint first place in this category. It was submitted by Karlis Freivalds and Paulis Kikusts (karlisf,paulis@cclu.lv), from the University of Latvia, Latvia. It was generated using the ActiveX Graphical Diagramming Engine (`http://www.gradetools.com/`). The structure of the graph was discovered using a variant of barycentric layout, and subsequent refinement of node and edge positions was done interactively.

The other first-place drawing (Figure 7) was submitted by Rowena Mankelow (c9514915@studentmailbox.newcastle.edu.au), from the University of Newcastle, Australia. An initial layout was computed using a force-directed method. Nodes were then repositioned by hand to emphasize important biochemical pathways, such as the tricarboxylic acid cycle.

2.4 Category D

The only requirement for submissions in this category is that they be some form of artistic expression inspired by or related to graph drawing. There were only two entries in this category, but both were deemed worthy of prizes.

Second place was awarded to Michael Goodrich (goodrich@jhu.edu) for the remarkable image of a seashell shown in Figure 8. The pattern on the shell, reminiscent of tree layouts, was generated naturally.

First place in Category D went to Ulrik Brandes (ub@cs.brown.edu), from Brown University. This drawing, called “Shiftset,” depicts the state-transition

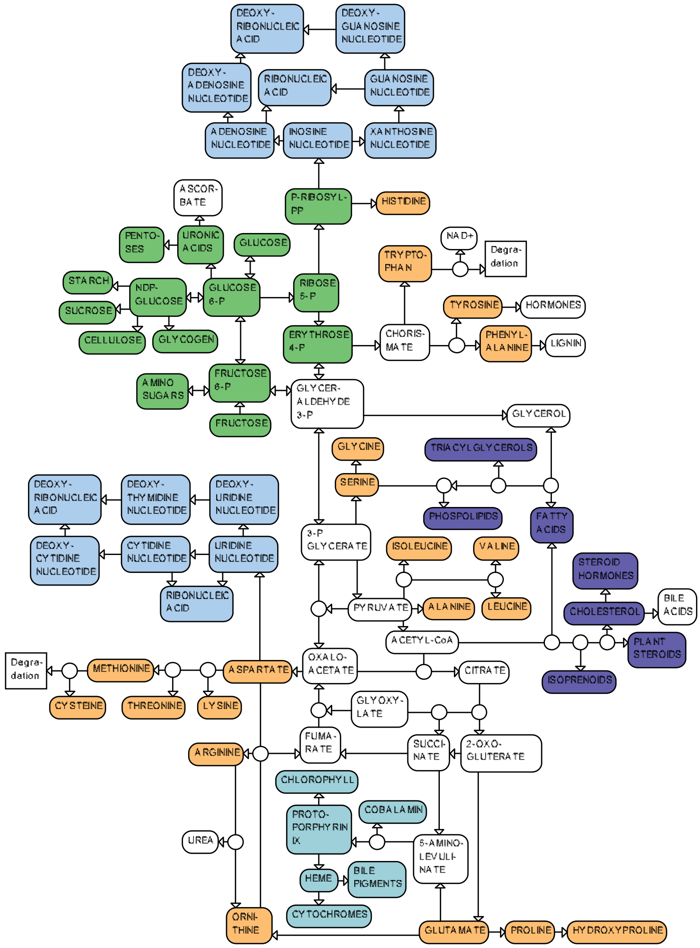


Fig. 6. Joint first place, Category C (original in color).

diagram of an eight-bit shift register, arranged circularly according to binary states. In the original, edges are colored to indicate the insertion of 0 (yellow) or 1 (orange), respectively. Further details can be found in Brandes's account of his Category B submission [1], on which this graph and drawing are based.

3 Observations and Conclusions

As in past years, most of the winners combined automated and manual techniques to great effect. Given this distinct pattern in how graph-drawing software is used, it is perhaps surprising that few systems have been designed to give

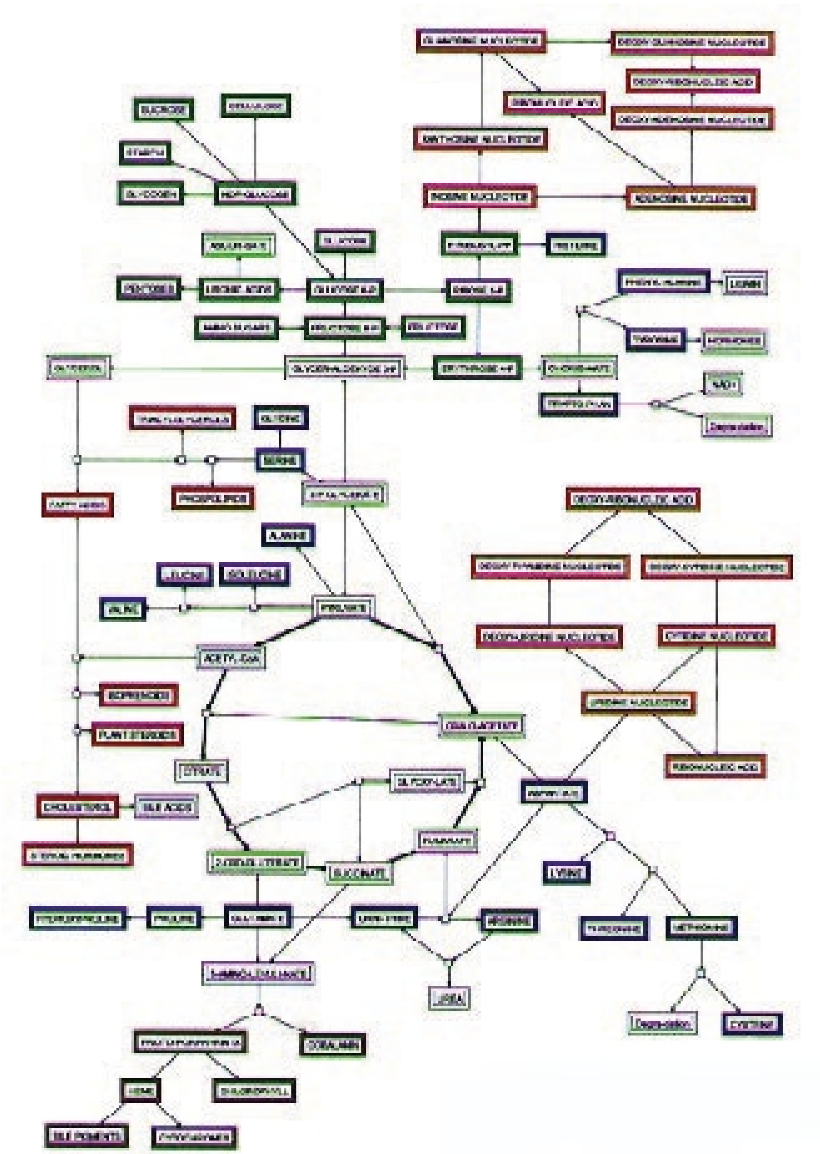


Fig. 7. Joint first place, Category C (original in color).

Fig. 8. Second place, Category D (original in color).

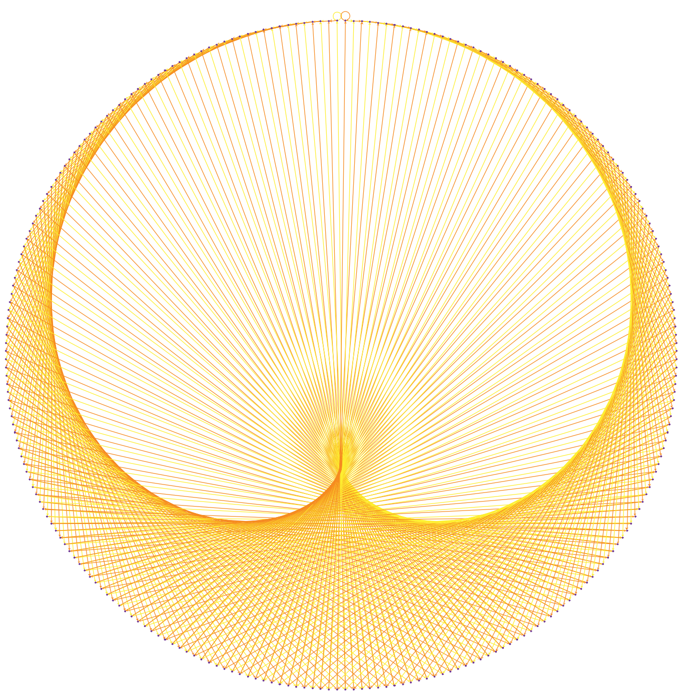


Fig. 9. First place, Category D (original in color).

explicit support to this kind of human-computer cooperative design. A future graph-drawing contest may therefore include an interactive-editing category.

Another trend that continued from previous years is that of using graph drawing as an analysis tool. No submission exemplifies this approach better than the first-place winner in Category B.

Category D, the artistic category, received fewer submissions this year, but made up in originality and beauty what was lacking in numbers.

Acknowledgments

Sponsorship for this contest was provided by AT&T Research, MERL–A Mitsubishi Electric Research Laboratory, and Tom Sawyer Software.

References

1. U. Brandes. Hunting down graph B. In this volume.
2. P. Eades and J. Marks. Graph-drawing contest report. In R. Tamassia and I. G. Tollis, editors, *Lecture Notes in Computer Science: 894 (Proceedings of the DIMACS International Workshop on Graph Drawing '94)*, pages 143–146, Berlin, October 1994. Springer.
3. P. Eades and J. Marks. Graph-drawing contest report. In F. J. Brandenburg, editor, *Lecture Notes in Computer Science: 1027 (Proceedings of the Symposium on Graph Drawing GD '95)*, pages 224–233, Berlin, September 1995. Springer.
4. P. Eades, J. Marks, P. Mutzel, and S. North. Graph-drawing contest report. In S. H. Whitesides, editor, *Lecture Notes in Computer Science: 1547 (Proceedings of the Symposium on Graph Drawing GD '98)*, pages 423–435, Berlin, August 1998. Springer.
5. P. Eades, J. Marks, and S. North. Graph-drawing contest report. In S. North, editor, *Lecture Notes in Computer Science: 1190 (Proceedings of the Symposium on Graph Drawing GD '96)*, pages 129–138, Berlin, September 1996. Springer.
6. P. Eades, J. Marks, and S. North. Graph-drawing contest report. In G. DiBattista, editor, *Lecture Notes in Computer Science: 1353 (Proceedings of the Symposium on Graph Drawing GD '97)*, pages 438–445, Berlin, September 1997. Springer.
7. `http://www.ms.mff.cuni.cz/acad/kam/conferences/GD99/contest/rules.-html`.
8. J. C. Huth, editor. *Das Lindenstrasse-Universum.* Köln, 1998.
9. G. Michal. *Biochemical Pathways.* Spektrum Akademischer Verlag, Heidelberg, 1999.
10. `http://vlado.fmf.uni-lj.si/pub/networks/pajek/`.

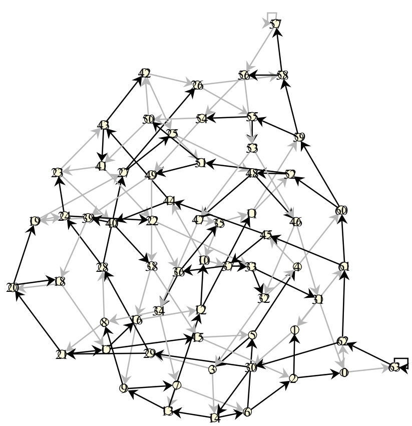
Hunting Down Graph B*

Ulrik Brandes

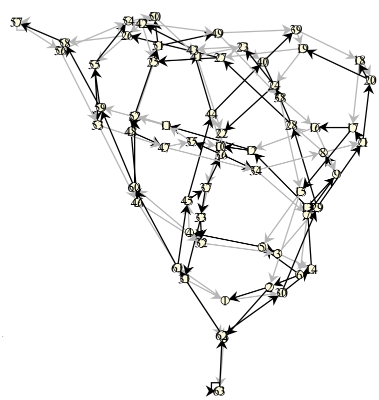
Department of Computer Science, Brown University
Providence, Rhode Island 02912-1910
Ulrik.Brandes@uni-konstanz.de

Abstract. Graph B, the "Theory Graph", of the 1999 Graph Drawing Contest is visually explored using available and some hand-written graph drawing software.

This year's "Theory Graph" was given in GML-format.[1] It is a directed graph, given without coordinates, but with a suspicious-looking vertex numbering (non-decreasing order, range 15070–15259, constant increment of 3) and edges colored either in green or in blue.[2] So let's use a simple spring embedder (Fruchterman/Reingold's variant [1] as implemented in LEDA's [2] graph editor GraphWin) to get a first impression of the graph's structural characteristics:



2D spring embedding



3D spring embedding

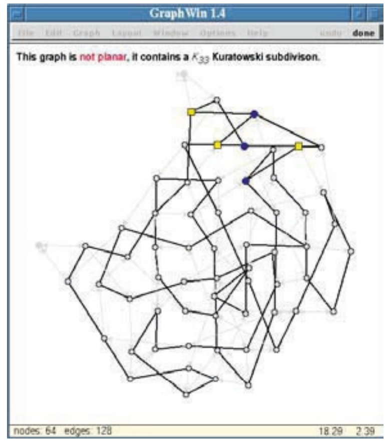
No revelation, though fairly symmetric in 3D. Inspection shows that the graph is sparse (64 vertices, 128 edges – hmm, powers of 2), but not planar.

* Die Arbeit wurde mit Unterstützung eines Stipendiums im Rahmen des Gemeinsamen Hochschulsonderprogramms III von Bund und Ländern über den DAAD ermöglicht.

1 GML homepage http://www.fmi.uni-passau.de/~himsolt/Graphlet/GML/.

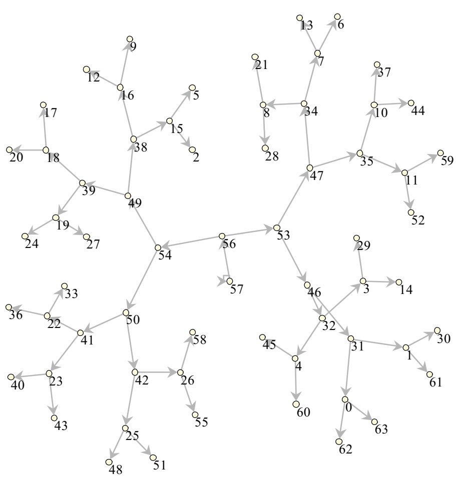
2 Here represented by gray and black, respectively.

J. Kratochvíl (Ed.): GD'99, LNCS 1731, pp. 410–415, 1999.

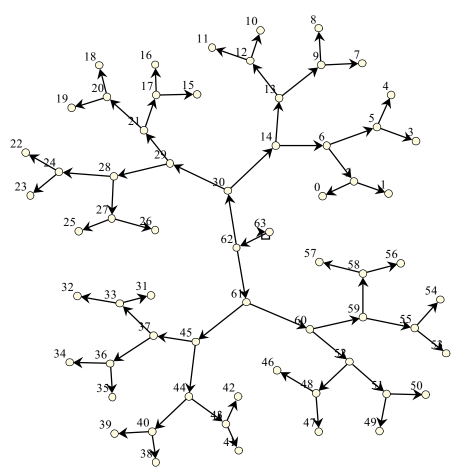


Moreover, it appears to be 4-regular, with each vertex incident to two incoming edges of different color, and two outgoing edges of the same color. There seem to be only two exceptions to this observation (due to a green and a blue loop).

Since it is to be expected that the competition graph is obfuscated, we simply index vertices according the order in which they are given (a default labeling option in GraphWin) and take a look at the green and blue edge-induced subgraphs:



subgraph induced by green edges

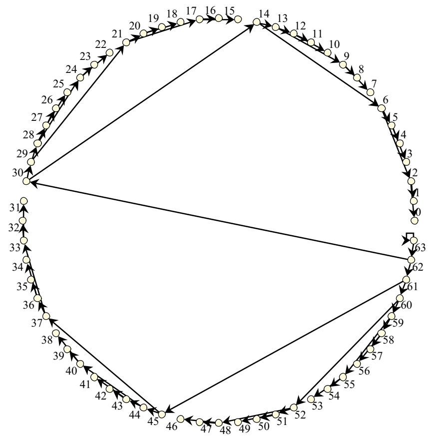


subgraph induced by blue edges

Surprise! The powers of two turn out not to be incidental. These layouts indeed show that the graph consists of two uni-colored complete binary trees with an additional parent of the root. And the order of the vertices was obviously produced by a post-order traversal of the blue tree. But why the additional vertex with a loop?

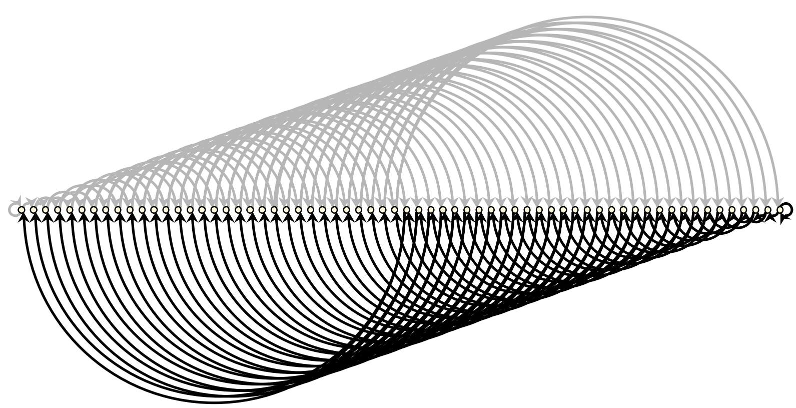
With a computer science background it is unavoidable to think of binary trees as being labeled with `0`/`1`-strings, where the left and right child of an internal node are labeled by appending `0` and `1`, respectively, to the parent's label. Now it

takes only modest intuition to replace the post-order numbering with its binary representation and to hypothesize that climbing down the tree could correspond to appending `0`s and `1`s, while dropping the leading digit. Sound familiar? Note how a circular layout of the blue tree spells



like shift register `;-)`. More serious evidence supporting this hypothesis is that internal nodes of one tree are leafs of the other, and vice versa. The numbering is not quite right (the binary representation of the green root's index should be `000000`, not `111010`), but reordering proves that this is just another obfuscation.[3]

We conclude that Graph B is the transition graph of a 6-bit shift register. Placing vertices equidistantly on a line according to the "appropriate" numbering (i.e. indices corresponding to bit sequences of register states) yields

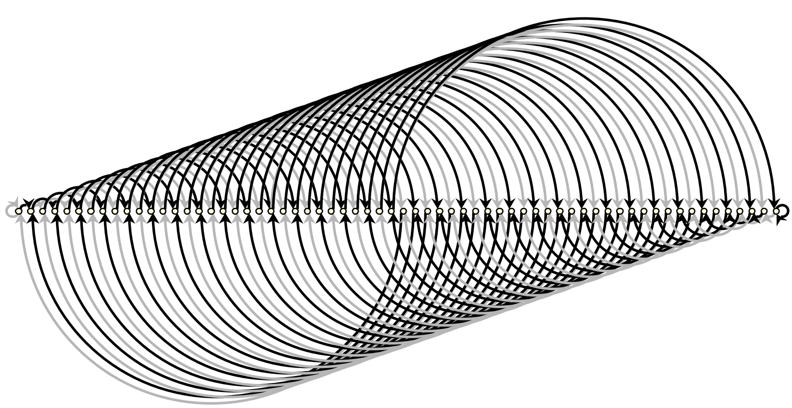


[3] The permutation is $\pi = (31, 30, 47, 29, 28, 46, 55, 27, 26, 45, 25, 24, 44, 54, 59, 23, 22, 43, 21, 20, 42, 53, 19, 18, 41, 17, 16, 40, 52, 58, 61, 15, 14, 39, 13, 12, 38, 51, 11, 10, 37, 9, 8, 36, 50, 57, 7, 6, 35, 5, 4, 34, 49, 3, 2, 33, 1, 0, 32, 48, 56, 60, 62, 63)$.

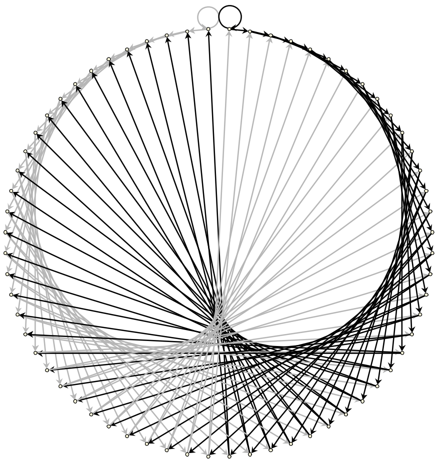
where edges are semicircles with radii half the distance between their incident vertices. Clearly, this drawing is not all that different from the suitably ordered adjacency matrix shown on the right.

```
 0: GG..............................................................
 1: ..GG............................................................
 2: ....GG..........................................................
 3: ......GG........................................................
 4: ........GG......................................................
 5: ..........GG....................................................
 6: ............GG..................................................
 7: ..............GG................................................
 8: ................GG..............................................
 9: ..................GG............................................
10: ....................GG..........................................
11: ......................GG........................................
12: ........................GG......................................
13: ..........................GG....................................
14: ............................GG..................................
15: ..............................GG................................
16: ................................GG..............................
17: ..................................GG............................
18: ....................................GG..........................
19: ......................................GG........................
20: ........................................GG......................
21: ..........................................GG....................
22: ............................................GG..................
23: ..............................................GG................
24: ................................................GG..............
25: ..................................................GG............
26: ....................................................GG..........
27: ......................................................GG........
28: ........................................................GG......
29: ..........................................................GG....
30: ............................................................GG..
31: ..............................................................GG
32: BB..............................................................
33: ..BB............................................................
34: ....BB..........................................................
35: ......BB........................................................
36: ........BB......................................................
37: ..........BB....................................................
38: ............BB..................................................
39: ..............BB................................................
40: ................BB..............................................
41: ..................BB............................................
42: ....................BB..........................................
43: ......................BB........................................
44: ........................BB......................................
45: ..........................BB....................................
46: ............................BB..................................
47: ..............................BB................................
48: ................................BB..............................
49: ..................................BB............................
50: ....................................BB..........................
51: ......................................BB........................
52: ........................................BB......................
53: ..........................................BB....................
54: ............................................BB..................
55: ..............................................BB................
56: ................................................BB..............
57: ..................................................BB............
58: ....................................................BB..........
59: ......................................................BB........
60: ........................................................BB......
61: ..........................................................BB....
62: ............................................................BB..
63: ..............................................................BB
```

Instead of the coloring edges according to membership in what are essentially the 6-step reachability trees from states `000000` and `111111`, we can use the colors to indicate whether `0` or `1` is inserted as least significant bit:

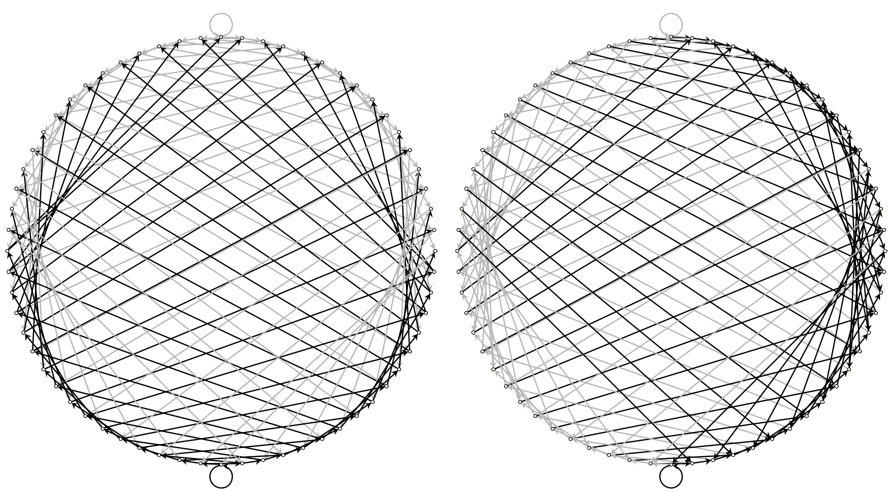


Circular layouts using this ordering are even nicer,



given coloring 0/1-insertion coloring

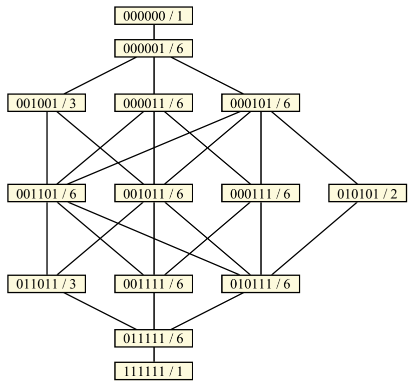
but less informative. If, however, we consider states to differ the most, if they differ in every single bit, a more useful convention could be to place 1-complements diametral:



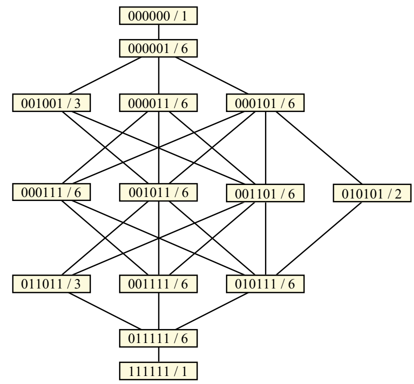
given coloring 0/1-insertion coloring

The drawing on the right clearly shows that even numbers form the left, while odd numbers constitute the right half of the circle. Finally, it would be interesting to display the remainders under cyclic shifting by placing the vertices of each remainder on a distinct circle. To place the circles, we first consider the graph obtained by contracting the vertices in each remainder and deleting parallel edges. Since the states in each remainder have an invariant number of `0` and `1`, there is a natural layering of the remainder states. By applying an instance of

the layered layout framework of Sugiyama et al. [4] (as implemented in the AGD library [3]) we obtain



with crossing minimization



manually rearranged to emphasize symmetry

where remainders are labeled with a representative state and the number of states contained. Note the striking resemblance of the initial 3D spring embedding! We leave it to the interested reader to determine a useful ordering of states when remainders are expanded into circles in three dimensions.

References

1. Thomas M.J. Fruchterman and Edward M. Reingold. Graph-drawing by force-directed placement. *Software—Practice and Experience*, 21(11):1129–1164, 1991.
2. Kurt Mehlhorn and Stefan Näher. *The* LEDA *Platform of Combinatorial and Geometric Computing.* Cambridge University Press, 1999. Project home page at `http://www.mpi-sb.mpg.de/LEDA/`.
3. Petra Mutzel, Carsten Gutwenger, Ralf Brockenauer, Sergej Fialko, Gunnar W. Klau, Michael Krüger, Thomas Ziegler, Stefan Näher, David Alberts, Dirk Ambras, Gunter Koch, Michael Jünger, Christoph Buchheim, and Sebastian Leipert. A library of algorithms for graph drawing. In Sue H. Whitesides, editor, *Proceedings of the 6th International Symposium on Graph Drawing (GD '98)*, volume 1547 of *Lecture Notes in Computer Science*, pages 456–457. Springer, 1998. Project home page at `http://www.mpi-sb.mpg.de/AGD/`.
4. Kozo Sugiyama, Shojiro Tagawa, and Mitsuhiko Toda. Methods for visual understanding of hierarchical system structures. *IEEE Transactions on Systems, Man and Cybernetics*, 11(2):109–125, 1981.

Orthogonal and Straight-Line Drawings of Graphs with Succinct Representations*

Ho-Lin Chen and Hsu-Chun Yen

Dept. of Electrical Engineering, National Taiwan University
Taipei, Taiwan, R. O. C.
yen@cc.ee.ntu.edu.tw

The concept of a *hierarchical design* has played an important role in various areas of computer science and engineering, including software engineering, CAD, among others. Graphs with hierarchical structures, which are capable of describing large-scale regular structures in a succinct manner, have naturally become an interesting and important modeling tool for facilitating such a hierarchical design. In the literature, a number of succinct models for representing graphs with hierarchical structures have been proposed. See, e.g., [4,5]. In each of such hierarchical graph models, a succinct description is capable of representing a graph (which can be thought of as the *expansion* of the description) whose size is exponential in the length of the description.

A *hierarchical graph* [4,5] $\Gamma = (G_1, ..., G_k)$ of *depth* k contains k *cells* $G_1, ..., G_k$, each of which is a graph consisting of two types of vertices, namely, *terminals* and *nonterminals*. Intuitively speaking, nonterminals are those that are going to be replaced by cells of smaller indices during the expansion process. Suppose $G_i = (V_i, E_i), 1 \leq i < k$, where V_i and E_i are the sets of vertices and edges of G_i, respectively. For each G_i, there is a *pin assignment function* $\rho_i : \{1, ..., d_i\} \rightarrow V_i$ (where d_i is a positive integer), which specifies the way vertices are connected to the upper layer. The d_i is called the *degree* of G_i, and each of the vertices in $\rho_i(\{1, ..., d_i\})$ is called a *frontier vertex* (*f-vertex*, for short). Each nonterminal inside G_i is specified as (m, G_j) where m (an integer) represents the unique *name* of the nonterminal, and G_j, $1 \leq j < i$, denotes the *type* of the nonterminal. A nonterminal of type G_j has degree d_j, and each incident edge is labeled by a unique integer in $\{1, ..., d_j\}$. The *expansion* of G_i, denoted by $E(G_i)$, is obtained by expanding all of its subcells $G_1, ..., G_{i-1}$ recursively, and then replacing each nonterminal of type G_j $(1 \leq j < i)$ inside G_i by a copy of $E(G_j)$ in such a way that each incident edge labeled l $(1 \leq l \leq d_j)$ is connected to the f-vertex $\rho_j(l)$. The *expansion* of a hierarchical graph $\Gamma = (G_1, ..., G_k)$, denoted by $E(\Gamma)$, is defined to be $E(G_k)$.

In this research, our goal is to 'draw' hierarchical planar graphs on *grids* in the styles of *straight-line drawing* (i.e., each edge is a straight-line segment) and *orthogonal drawing* (i.e., each edge is a chain of horizontal and vertical segments) [2]. Since the size of the expansion of a hierarchical graph can be exponential in the length of its succinct representation, care must be taken when defining

* This work was supported in part by the National Science Council of the Republic of China under Grant NSC-88-2815-C-002-002-E.

J. Kratochvíl (Ed.): GD'99, LNCS 1731, pp. 416–417, 1999.

the output of a graph drawing algorithm. In our setting, the outputs of our drawing algorithms are succinct representations of drawings and not of drawings themselves. (That is, we require that the drawing of a hierarchical graph be expressible succinctly as well.) To this end, the drawings of two copies of the same cell must be 'identical' in the sense that one drawing can be obtained by the operations of *flipping* with respect to the y-axis and/or *rotations* of $90^o, 180^o$ or 270^o from the other. A hierarchical graph is said to have a *planar* straight-line (resp., orthogonal) grid drawing if its expansion can be drawn on grids in a straight-line (resp., orthogonal) fashion without edge crossings subject to the above constraints regarding copies of each cell. It should be noted that our *hierarchical graph model* differs from that of 'hierarchical graphs' used in, e.g., [3]. The latter refers to graphs whose vertices are assigned to layers, and the so-called 'hierarchical drawing' is to place all the vertices of a hierarchical graph on a set of equally-spaced horizontal lines.

The main contributions of this paper include the design and analysis of orthogonal and straight-line drawing algorithms which operate on the succinct descriptions of hierarchical graphs directly, and output succinct representations of drawings. (For related results, the reader is referred to [1].) We do not require that the hierarchical graphs be expanded in order for our algorithms to work. To the best of our knowledge, conventional graph drawing algorithms operate only on completely specified graphs. For hierarchical graphs $(G_1, ..., G_k)$ with the number of outgoing connections in each G_i bounded by 2, we derive a planar straight-line grid drawing algorithm which runs in time $\sum_{i=1}^{k} |G_i|$, where $|G_i|$ denotes the number of vertices in G_i. (Notice that $\sum_{i=1}^{k} |G_i|$ is linear in the size of the succinct representation.) The drawing area of the expanded graph is bounded by $O(2^{3k} \prod_{i=1}^{k}(|G_i|-2)^6)$. For orthogonal grid drawings of hierarchical planar graphs, we present an algorithm which runs in $O(\sum_{i=1}^{k} |G_i|^2)$ time, provided that the input graph is 2-connected (i.e., a graph which remains connected even if a single vertex is removed). The drawing area and the total number of bends of the expanded graph are bounded by $O(n^2)$ and $(max_{i=1,...,k}\{|G_i|\})^k$, respectively, where n is the number of vertices in the expanded graph. Our algorithm can also be used to report whether the input graph exhibits a planar straight-line (or orthogonal) grid drawing.

References

1. F. Brandenburg, Designing Graph Drawings by Layout Graph Grammars, *Graph Drawing'94*, LNCS 894, pp. 416-427, 1994.
2. G. Di Battista, P. Eades, R. Tamassia, and I. Tollis, *Graph Drawing: Algorithms for the Visualization of Graphs,* Prentice-Hall, 1999.
3. P. Eades, X. Lin, and R. Tamassia, An Algorithm for Drawing a Hierarchical Graph, *Internat. J. Comput. Geom. Appl.*, Vol. 6, pp. 145-156, 1996.
4. T. Lengauer, Efficient Algorithms for Finding Minimum Spanning Forests of Hierarchically Defined Graphs, *J. Algorithms*, Vol. 8, pp. 260-284, 1987.
5. T. Lengauer, Hierarchical Planarity Testing Algorithms, *J. ACM*, Vol. 36, No. 3, pp. 474-509, 1989.

Electronic Biochemical Pathways*

Carl-Christian Kanne[1], Falk Schreiber[2], and Dietrich Trümbach[3]

[1] Lehrstuhl für Praktische Informatik III, Prof. Guido Moerkotte,
Universität Mannheim, D7, 27, 68131 Mannheim, Germany,
[2] Lehrstuhl für Informatik, Prof. Franz J. Brandenburg,
Universität Passau, 94030 Passau, Germany,
[3] Institut für Organische Chemie, Prof. Johann Gasteiger,
Universität Erlangen-Nürnberg, Nägelsbachstr. 25, 91052 Erlangen, Germany,

The biochemistry of living beings is a complex network of reactants, products and enzymes with multiple interconnections representing reactions and regulation. Examples are given by the Boehringer poster [5] and the atlas "Biochemical Pathways" [6]. In the *Electronic Biochemical Pathways* project we intend to build a platform for a convenient electronic access to the information covered by the poster and the atlas.

The *Electronic Biochemical Pathways* has three major parts. First, chemical information is modeled accurately and in detail. This data is stored in a database which also contains explicit connections between the different subjects (for example, showing the diseases related to a certain reaction pathway, see also [4]). Using this database and advanced automatic visualization of the involved reaction graphs, the resulting software system will provide users with new ways of considering and analyzing metabolic pathways.

The project is divided into three parts: Acquiring and modeling the data, the suitable database environment, and the graph visualization system.

Acquiring and modeling the data

This part deals specifically with the molecule- and reaction-attributes:

- Code all chemical structures from the Boehringer poster (part 1) on the atomic level. Thereafter, we can implement a full structure and a substructure search [8].
- Registration of the compound-names in German and English both in HTML format.
- Assignment of the compounds to (co-)reactant and (co-)product and the classification of these compounds into chemical groups.
- Specification of the conditions for a reaction (enzymes, regulation, environment). Detailed information about the reaction-mechanisms by specifying a mapping of product atoms to reactant atoms.

This core data from the poster is enriched by structural information, like grouping of relation reactions into standard reaction chains (pathways like glycolysis or the citrate cycle), layout constraints for reaction graphs, amino acid and gene sequences for proteins, and links into specific web databases.

* This work was supported by the German Ministry of Education and Research (BMBF)

J. Kratochvíl (Ed.): GD'99, LNCS 1731, pp. 418–419, 1999.

Building a suitable database environment

The above-mentioned concepts belong to a wide range of different abstraction levels. They were mapped into a complex, object-oriented, conceptual model using UML [7]. The data model was implemented using a commercial database management system. It is the base component of the *Electronic Biochemical Pathways.*

Special consideration was given to the addition, export and interaction with external data. A common exchange format for chemical and biochemical data is XML (eXtensible Markup Language) [1]. Instead of multiple conversion programs, a specialized XML database system is being developed which stores all the relevant project's information directly in XML and will replace the commercial database server in the deployment version of our system.

Creating a graph visualization system

Existing graph drawing algorithms [2,3] are not sufficient to represent pathways according to the conventions of biochemistry. As a consequence, in this part of the project we are developing algorithms tailored to biochemical pathways.

The capabilities of automatic drawing go far beyond the ones shown by the poster: diagrams of pathways can be combined with other pieces of information (like cell compartments or membrane transport processes) or similar pathways of different organisms can be compared with each other by overlying identical parts and emphasizing different parts. One can use graphic features such as highlighting and animation. The automatic generation of pathway diagrams offers new chances for the navigation through pathways like different resolution steps from general pathway maps to diagrams of all details of a single pathway. Pathways can be expanded dynamically and drawings of new parts can be produced on demand. It is possible to connect diagrams with each other, for example pathways and the tree of the enzyme classification of the corresponding enzymes.

Forthcoming information is available under http://www.biochemical-pathways.de.

References

1. T. Bray, J. Paoli, and C. M. Sperberg-McQueen. Extensible Markup Language (XML) 1.0 W3C Recommendation 10-Feb.-98. http://www.w3.org/TR/REC-xml.
2. G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. Algorithms for Drawing Graphs: An Annotated Bibliography. *Comput. Geom.*, 4(5):235–282, 1994.
3. G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. *Graph drawing: algorithms for the visualization of graphs.* Prentice Hall, New Jersey, 1999.
4. P. W. Erhardt. *Drug Metabolism - Databases and High-Throughput Testing During Drug Design and Development.* Blackwell Science, Oxford, 1999.
5. G. Michal. Biochemical Pathways (Poster), 1993. Boehringer Mannheim GmbH.
6. G. Michal. *Biochemical Pathways.* Spektrum Akadem. Verlag, Heidelberg, 1999.
7. J. Rumbaugh, I. Jacobson, and G. Booch. *The Unified Modeling Language Reference Manual.* Addison-Wesley, 1998.
8. R. E. Stobaugh. Chemical Substructure Searching. *J. Chem. Inf. Comput. Sci.*, 25:271–275, 1985.

Author Index